ADST-93-W003015 28 May 1993

1

Advanced Distributed Simulation Technology

Distributed Interactive Simulation Protocol Extensions

CONTRACT NO. N61339-91-D-001
CDRL SEQUENCE NO. A00A

Prepared for:

US Army
Simulation, Training, and Instrumentation Command
12350 Research Parkway
Orlando, FL 32826

Prepared by:

LORAL
Systems Company

ADST Program Office
12433 Research Parkway Suite 303
Orlando, FL 32826

ADST-93-W003015 28 May 1993

REPORT DOCUMENTATION PAGE

Form approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	5/28/93	Technical Report

4. TITLE AND SUBTITLE
Distributed Interactive Simulation Protocol Extensions

5. FUNDING NUMBERS
C N61339-91-D-0001
CDRL A00A

6. AUTHOR(S)
O'Brien, Sheila

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
BDM Federal, Inc.
ADST Program Office
12443 Research Parkway, Suite 303
Orlando, FL 32826

8. PERFORMING ORGANIZATION REPORT NUMBER
ADST/TR--93-003015

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Simulator Training and Instrumentation Command (STRICOM)
12350 Research Parkway
Orlando, FL 32826-3275

Loral Systems Company
ADST Program Office
12443 Research Parkway
Suite 303
Orlando, FL 32826

10. SPONSORING ORGANIZATION REPORT

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The purpose of this document is to report on the on going changes and extensions to the DIS and SIMNET protocols which are a result of ADST-related activity and where possible activities outside the ADST contract. This report reflects the ADST Program Management Office's effort to provide a central point of coordination and technical oversight for all proposed PDUs and PDU changes. This is an effort to ensure that all PDUs developed and proposed have the widest possible applicability in the DIS community as well as ensuring no duplication of effort.

14. SUBJECT TERMS

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	17. SECURITY CLASSIFICATION OF THIS PAGE	17. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std Z39-18
298-102

TABLE OF CONTENTS

LIST OF TABLES AND FIGURES

1 INTRODUCTION

The purpose of this document is to report on the on going changes and extensions to the DIS and SIMNET application protocols which are a result of ADST-related activity and where possible activities outside the ADST contract. This report reflects the ADST Program Management Office's effort to provide a central point of coordination and technical oversight for all proposed PDUs and PDU changes. This is an effort to ensure that all PDUs developed and proposed have the widest possible applicability in the DIS community as well as ensuring no duplication of effort.

2 SCOPE

This document contains a revision to the Advanced Distributed Simulation Technology (ADST) DIS and SIMNET Protocol Extensions Summary. It is organized into five sections and three appendixes including (1) Introduction, (2) Scope, (3) Applicable Document, (4) Summary of Protocol Extensions for ADST Activities, (5) Summary of Protocol Extensions for Other Activities, Appendix A: Protocol Extension Template, Appendix B: DIS Protocol Extensions, and Appendix C SIMNET Protocol Extensions.

The information provided for each Delivery Order (DO) is related to protocol extensions only. For a more complete summary of each DO, see the *Advanced Distributed Simulation Technology (ADST) Delivery Order Summary Handbook* [Loral 1993a].

This version is a summary of the continuing survey of all ADST LSE and Delivery Order activities that are proposing and making extensions to either the DIS or SIMNET protocol. This version also includes a summary of the protocol related activities of the 8th workshop on Standards for the Interoperability of Defense Simulations.

2.1. PROTOCOL EXTENSION PROCESS

Previous versions of this report were titled "Candidate DIS and SIMNET Protocol Extensions". The title has been changed to reflect the evolving character of the document. This version of the document has been reorganized so that this document can support the configuration management of protocol extensions. The body of the document is a summary of ADST and non-ADST protocol activities and the appendices contain complete documentation of each protocol extensions that have been implemented by ADST DO and LSE activities.

Appendix A defines a template for documenting protocol extensions. Protocol extensions undertaken by the ADST program will be documented using this format and added to appendix B or C of this document.

3 APPLICABLE DOCUMENTS

Loral, 1993a	ADST Delivery Order Summaries: Dec 1992 - Feb 1993, Loral Systems Company, February 1993.
Loral, 1993b	Distributed Interactive Simulation Architecture Description Document, Loral Systems Company, ADST/TR--93-003010, 28 May 1993.
IEEE, 1993	1278-1993, Standard for Information Technology, Protocols for Distributed Interactive Simulation Applications, IEEE, New York, NY, March 1993
IST, 1992a	Military Standard (Final Draft): Protocol Data Units for Entity Information and Entity Interaction in a Distributed Interactive Simulation, IST-PD-91-1, UCF Institute for Simulation and Training, May 8, 1992.
IST, 1992b	Summary Report, The Seventh Workshop on Standards for the Interoperability of Defense Simulations, IST-CF-92-17, UCF Institute for Simulation and Training, September 1992.
IST, 1993a	Proposed IEEE Standard Draft: Standard for Information Technology - Protocols for Distributed Interactive Simulation Applications, Version 2.0 - Second Draft, IST-CR-93-01, UCF Institute for Simulation and Training, 22 March 1993.
IST, 1993b	Enumeration and Bit Encoded Values for Use with Protocols for Distributed Interactive Simulation Applications, Version 2.0 - Second Draft, IST-CR-93-02, UCF Institute for Simulation and Training, 22 March 1993.
IST, 1993c	Operational Concept for Distributed Interactive Simulation Draft 2.2, IST-TR-93-10, UCF Institute for Simulation & Training, March 1993.
IST, 1993d	Standards Development Guidance Document for Distributed Interactive Simulation Standards Development Draft 2.1, IST-TR-93-11, UCF Institute for Simulation & Training, March 1993.
IST, 1993e	Proposed IEEE Draft Standard: Communication Architecture for Distributed Interactive Simulation (CADIS), IST-CR-93-07, UCF Institute for Simulation & Training, March 1993.
IST, 1993f	Rational Document for Proposed IEEE Draft Standard: Communication Architecture for Distributed Interactive Simulation (CADIS), IST-CR-93-08, UCF Institute for Simulation & Training, March 1993.

IST, 1993g	Proposed IEEE Draft Standard: Exercise Control and Performance Measures Feedback Requirements for Distributed Interactive Simulation, IST-CR-93-05, UCF Institute for Simulation & Training, March 1993.
IST, 1993h	Rational Document for Proposed IEEE Draft Standard: Exercise Control and Performance Measures Feedback Requirements for Distributed Interactive Simulation, IST-CR-93-06, UCF Institute for Simulation & Training, March 1993.
IST, 1993i	Proposed IEEE Draft Standard: Fidelity Description Requirements for Distributed Interactive Simulation, IST-CR-93-04, UCF Institute for Simulation & Training, March 1993.
IST, 1993k	Summary Report, The Eighth Workshop on Standards for the Interoperability of Defense Simulations, IST-CF-93-10, UCF Institute for Simulation and Training, March 1992.
BBN, 1991	Arthur Pope, Richard L. Schaffer. The SIMNET Network and Protocols BBN Report Number 7627, June 1991.

4 SUMMARY OF ADST ACTIVITIES

4.1. DELIVERY ORDERS

The following sections are an enumeration of the ADST program's delivery orders (DOs) and potential DOs; a discussion of the impact, if any, that the DOs will have on protocol standards. Where applicable, the impact, if any, on databases will also be reported.

4.1.1. Active Delivery Orders

4.1.1.1. AIRNET AeroModel & Weapons Model Conversion

The AirNet AeroModel and Weapons Model Delivery Order provides specific enhancements to the eight existing Rotary Wing Aircraft (RWA) devices at the Ft. Rucker Aviation Testbed (AVTB). Increased Management Command and Control (MCC) subsystem functionality will provide additional tables and menus for supply and initialization of the RAH-66 Comanche. A new digital communication function will be added, allowing messages to be sent to/from the Tactical Operations Center (TOC), Fire Support Element (FSE), or Battlemaster and the RWA devices. The existing flight and weapons models will be replaced with tunable models utilizing tables for defining aircraft and weapons performance characteristics.

This effort has developed a new Digital Message Communications protocol extension. This extension was developed in conjunction with the DIS 2.0 Protocol Translator Gateway, developed under the CSRDF - BDS-D Interface DO, see 4.1.1.5. This collaboration allows the DMC protocol extension to be used within a SIMNET and/or DIS 2.0 (Draft) environment. This DO has implemented DIS 2.0 (Draft) Signal PDU's to carry tactical messages. This protocol extension is documented in Appendix B2.

4.1.1.2. BDS-D Architecture Definition & DIS Standards Development

This activity has implemented no changes to the DIS or SIMNET Protocols.

4.1.1.3. Battlefield Synchronization Demonstration

This activity will implement no changes to the DIS or SIMNET Protocols. This activity will use the CVCC extensions of the SIMNET protocol that are described in Appendix C4.

4.1.1.4. CGF Architecture & Integration of Higher Order Models

This activity will be a primary source for enhancements to the DIS protocol. This effort will propose a C3I protocol and other enhancements to support the expanding scope of the BDS-D virtual battlespace. These protocol enhancements are described in Appendix B1.

4.1.1.5. CSRDF - BDS-D Interface

At this time, this activity has not proposed any changes to the DIS protocol. This activity will produce a protocol translator capability that will allow the CSRDF system running DIS 1.0 Protocol to interoperate with the Ft. Rucker Simulators running the older SIMNET 6.6.1 Protocol. This effort will be a potential source of protocol changes in the future. This effort is utilizing the DMC protocol extension developed under the AIRNET AeroModel & Weapons Model Conversion DO (see 4.1.1.1).

4.1.1.6. CVCC '93 Tests

This activity will implement no changes to the DIS or SIMNET Protocols. This activity will use the CVCC extensions of the SIMNET protocol that are described in Appendix C4.

4.1.1.7. DOTD Training Delivery Order

This activity will implement no changes to the DIS or SIMNET Protocols.

4.1.1.8. Jayhawk Thunder

At this time, this activity has not proposed any changes to the DIS protocol. This activity will use the C3I extensions of the DIS protocol that are described in Appendix B1.

4.1.1.9. BDS-D System Definition Support

This activity will implement no changes to the DIS or SIMNET Protocols.

4.1.1.10. ModSAF

At this time, this activity has introduced no changes to the DIS or SIMNET Protocols. This effort may propose changes to the SIMNET protocol in the future and is scheduled to be DIS compliant. The Persistent Object Protocol that is used by SIMNET SAFOR and ModSAF is documented in Appendix C6.

4.1.1.11. MultiRad / War Breaker

This effort provides for networked extensions to Air Force Weapon systems as part of the networked Virtual Battlespace environment. Elements represented include fixed wing, F-16 and F-15, Unmanned Air Vehicles (UAV), JSTARS and Airborne Radar AWACS. The on-going Network Interface Unit (NIU) development is particularly important in linking non-SIMNET systems to the SIMNET Network as well as interfacing dissimilar simulation fidelity simulators. The NIU will serve as an important prototype Cell Adapter Unit (CAU) as defined in the DIS Architecture Document [Loral, 1993b]. The linking of existing simulation assets utilizing NIU/CAU capabilities is critical for affordable simulation network extension.

The SIMNET protocol extensions for this effort are further described in Appendix C5.

4.1.1.12. Non-Line of Sight, Phase 2

This activity will implement no changes to the DIS or SIMNET Protocols.

4.1.1.13. X-Rod (Experimental Anti-Tank Missile)

This activity will implement no changes to the DIS or SIMNET Protocols.

4.1.1.14. Vehicle Integrated Defense Systems

The feasibility analysis performed under this delivery order recommended the following changes to the SIMNET protocol: 1) create a new PDU to communicate the presence of a non-tangible field (e.g. laser beams, acoustic signatures, smoke clouds, etc.) 2) modify existing PDUs where necessary to increase descriptive fields adding unique VIDS simulation data. This feasibility analysis was performed to provide a design approach to conduct simulated threat engagements using electronic survivability suites. These threat engagements employ simulated sensor and countermeasure systems to provide measurements of survivability effectiveness for various tactics, techniques, and procedures used in conjunction with different configurations of the electronic survivability suites. This simulation involves magnetic, optical, acoustic, and amorphous fields which are either poorly covered or not covered at all by the present SIMNET. See sections 5.7 - 5.9.

The approach consists of modifying manned M1 simulators to add Missile Countermeasures Device (MCD), Laser Warning Receiver (LWR), Missile Warning System (MWS), Multi-Salvo Grenade Launcher (MSGL) and counter fire models, controlled by emulated Threat Resolution Model (TRM) software and a PC-based touch screen implementation of the VIDS Commander's Control and Display Console (CCDP). The approach includes simulation of rapid turret slewing as well as support for manual, semi-automatic and automatic VIDS operational modes. The SAFOR system is also being modified to generate new threat platforms and threats for simulated tactical combat engagements against the VIDS-equipped M1s on the Hunter-Ligget database. New SIMNET PDUs and smoke models are being developed in support of this approach.

The SIMNET protocol extension is documented in Appendix C6.

4.1.2. Potential Delivery Orders

4.1.2.1. BDS-D Testbed

This potential delivery order will provide a test bed for testing protocol extensions and will also be a source of protocol extensions.

4.1.2.2. Directorate of Simulation Training Development Test

At this time, this activity is not expected to change the DIS or SIMNET Protocols.

4.1.2.3. Project Sword

At this time, this activity is not expected to change the DIS or SIMNET Protocols.

4.1.2.4. Prairie Warrior

At this time, this activity is not expected to change the DIS or SIMNET Protocols.

4.1.2.5. Louisiana Maneuvers 94

At this time, this activity is not expected to change the DIS or SIMNET Protocols.

4.1.2.6. Jayhawk Thunder II

This potential delivery order will extended the C3I protocol extension developed under Jayhawk Thunder, see Appendix B1.

4.1.2.7. Project Stingray

This potential DO could impact the DIS or SIMNET protocols.

4.1.2.8. Anti-Armor ATD

At this time, this activity is not expected to change the DIS or SIMNET protocols.

4.1.3. Completed Delivery Orders

4.1.3.1. Hollis Experiment

In support of the Hollis Experiment, enhancements were made to the SIMNET data collection protocol, see Appendix C2.

4.1.3.2. Land Systems Future Battlefield

This activity implemented no changes to the DIS or SIMNET Protocols.

4.1.3.3. Leavenworth Node

This activity implemented no changes to the DIS or SIMNET Protocols.

4.1.3.4. Seamless Simulation Experiment

This activity implemented no changes to the DIS or SIMNET Protocols.

4.1.3.5. Smart Minefield Simulator

This activity has defined the Smart Mines Simulation Protocol to enhance the SIMNET Protocols. This Protocol is described in Appendix C1.

4.1.3.6. CVCC Battalion Formative Evaluation

This activity implemented no changes to the DIS or SIMNET Protocols. This activity used the CVCC extensions of the SIMNET protocol that are described in Appendix C4.

4.2. LSE TASKS

The following sections are an enumeration of the ADST program's Laboratory Sustainment Effort, LSE, activities, and potential LSE activities; a discussion of

the impact, if any, that the activity will have on protocol standards; and where applicable the impact, if any, on databases.

4.2.1. Active LSE Tasks

4.2.1.1. LSE Systems Engineering

This activity has implemented no changes to the DIS or SIMNET Protocols. But is a source of guidance on protocol and database development.

4.2.1.2. M1A2 Training Developments

This activity has implemented no changes to the DIS or SIMNET Protocols.

4.2.1.3. Electronic Information Exchange Network

This activity has implemented no changes to the DIS or SIMNET Protocols.

4.2.1.4. Software Support/Configuration Management

This activity has implemented no changes to the DIS or SIMNET Protocols.

4.2.1.5. Line of Sight, Anti-Tank

This activity has implemented no changes to the DIS or SIMNET Protocols.

4.2.1.6. HEL Intelligibility Study

This activity has implemented no changes to the DIS or SIMNET Protocols.

4.2.1.7. Software Development Facility

This activity has implemented no changes to the DIS or SIMNET Protocols.

4.2.2. Potential LSE Tasks

NA

4.2.3. Completed LSE Tasks

4.2.3.1. ATAC II

In support of the Air to Air Combat II (ATAC II) Delivery Order, a Missile Server was added to the network to allow the firing of Hellfire missiles with a remote designator. Previously, missile flyout was limited by the 7 km range limitation of the RWA device. The Missile Server handles missile flyout and enhances intervisibility calculations. These enhancements were implemented using the Missile Server Protocol documented in Appendix C3.

5 SUMMARY OF RELATED NON-ADST ACTIVITIES

5.1. DIS STANDARDS PROCESS

In an effort to expand the use of distributed interactive simulation technology, the DIS standards process was organized to develop industry-wide standards for distributed simulation. The First Workshop on the Interoperability of Defense Simulations was held in August of 1989. At this first DIS workshop, it was decided to use the SIMNET protocols as the basis for development of the initial DIS standard which would define the protocol for exchanging messages between simulation applications. Workshops have been held biennially since that time and have lead to the formal adoption in March 1993 of the *Standard for Information Technology - Protocols for Distributed Interactive Simulation Applications* [IEEE, 1993] as an Institute of Electrical and Electronics Engineers (IEEE) standard. The DIS Workshops and the overall standards effort are coordinated by the University of Central Florida's Institute for Simulation and Training with funding initially from DARPA and currently from STRICOM and DMSO.

The DIS Workshops are continuing the development of Interoperability standards for defense simulation. The DIS application protocols standard is the first of the DIS standards to be formally adopted. Development is underway on the following standards:

- Communication Architecture for Distributed Interactive Simulation (CADIS) [IST, 1993e]
- Fidelity Description Requirements [IST, 1993i]
- Exercise Control and Performance Measures Feedback Requirements [IST, 1993g]
- Field Instrumentation

Other potential standards include:

- DIS Architecture
- Common Database Standard

For further information on the standards process see the Standards Development Guidance Document for Distributed Interactive Simulation Standards Development [IST, 1993d].

As Version 1.0 of the standard began the process of the becoming an IEEE standard, the working groups began working on Version 2.0. This version will incorporate Simulation Management, Emissions capabilities, and Radio Communications. The final draft of this version will be approved by the working groups and begin the process of becoming an IEEE standard. Work will then begin on Version 3.0 which will continue to expand the depth and breadth of the virtual battlespace. Figure 1 is a summary schedule for the standardization of Protocols for DIS Applications.

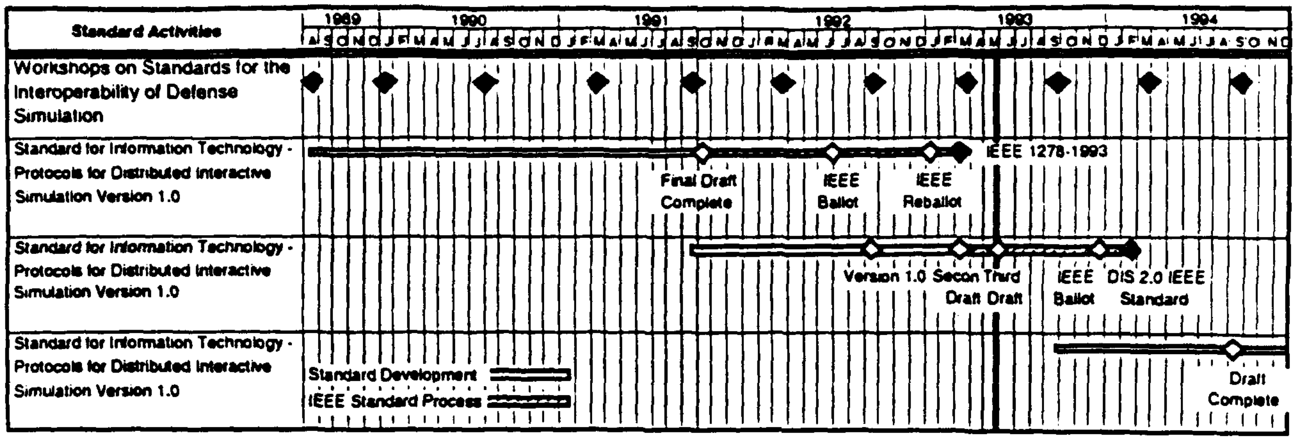


Figure 1. Standardization Activity for the DIS Application Protocol Standard.

Table 1 is a summary of the current and future capabilities of the protocol.

Table 1. Distributed Interactive Simulation Current and Future Capabilities.

FUNCTION	IMPLEMENTATION	SIMNET 6.6.1	DIS 1.0	DIS 2.0	DIS 3.0
Entity Appearance	Appearance PDU	Y			
	EntityStatePDU(1)		Y	Y	Y
Weapons					
DirectFire	FirePDU	Y	Y	Y	Y
IndirectFire	IndirectFirePDU	Y			
	FirePDU(2)		Y	Y	Y
Detonation	ImpactPDU	Y			
	DetonationPDU(3)		Y	Y	Y
LogisticsSupport	ServiceRequestPDU	Y	Y	Y	Y
	ResupplyOfferPDU	Y	Y	Y	Y
	ResupplyReceivedPDU	Y	Y	Y	Y
	ResupplyCancelPDU	Y	Y	Y	Y
	RepairCompletePDU	Y	Y	Y	Y
	RepairResponsePDU	Y	Y	Y	Y
Collisions	CollisionPDU	Y	Y	Y	Y
Electromagnetic	Emissions				
Radar	RadiatePDU	Y			
	EmitterPDU(4)			Y	Y
Communications	SINCGARSPDU	Y			
	SignalPDU			Y	Y
	TransmitterPDU			Y	Y
ECM	EmitterPDU			Y	Y
Laser	LaserPDU			Y	Y
Infrared	EmitterPDU				Y
Acoustic Emissions.	EmitterPDU				Y
EntityControl	RequestControlPDU				Y
Aggregation/Deaggregation	Aggregate PDU				Y
Exercise Management					
Initiation	Create Entity /Set Data PDUs			Y	Y
Start	Start/Resume PDUs			Y	Y
Resume	Start/Resume PDU			Y	Y
Ex. Termination	Stop/Freeze PDU			Y	Y
Freeze	Stop/Freeze PDU			Y	Y
Remove Entity	Remove PDU			Y	Y
Regenerate Entity	Set Data-Start PDUs			Y	Y
Observed Event	Event PDU			Y	Y
Parameter. Query	Data Query-Data PDUs			Y	Y
Action Req.	Action Request-Response PDUs			Y	Y
Message Log	Message PDU			Y	Y
Post Exercise Feedback					
Replay Exercise					Y
Control Replay Speed					Y
Control Origin and Magnify					Y
Control Groups of Entities					Y
Jump To Specified Event or Time					Y
Display Timeline					Y

Notes:
(1) Essentially same information, different parameters.
(2) Includes functions of SIMNET Indirect Fire PDU.
(3) Same Information As Impact PDU plus additional parameters.
(4) Includes Radar functions.

5.1.1. 8th Workshop

The 8th Workshop on the Interoperability of Defense Simulations was held in March of 1993. The status of the various standards documents is summarized in the Table 2.

Table 2. Status of Standards Documents

DOCUMENT	STATUS
DIS Protocol Standard Version 2.0	Second Draft
CASS Version 1.0	Final Draft
ECFR Version 1.0	Final Draft
FDR Version 1.0	First Draft
Environment Documents	Initial Drafts
Field Instrumentation	Initial Drafts

5.1.1.1. IEEE Standards Progress

The results of the IEEE P1278 second ballot were reported at the 8th Workshop.

- 86% of Ballots Returned
- 95% Voted Positive
- 3 Negative Votes
- 11 Positive Votes had Comments

As a result of the second ballot, the Standard for Information Technology, Prot..:ols for Distributed Interactive Simulation Applications was accepted as IEEE .278-1993.

5.1.2. 7th Workshop

The 7th Workshop on the Interoperability of Defense Simulations was held in September of 1992.

5.1.2.1. Discussion of potential changes to DIS Protocol

5.1.2.1.1. ENTITY STATE PDU

A recommendation was made to change the definition of the Entity Coordinate Vector. The current strict interpretation of the DIS standard results in a dynamic bounding volume. The reason the bounding volume moves is because articulated parts can change the shape of the bounding volume. Changing the shape of the bounding volume serves no useful purpose and adds a computational burden to a simulation. The addition of a sentence which states, "A bounding volume is defined as the cube which includes the entity without articu' ed parts." will make the entity coordinate system static. (See position paper 9.-30).

5.1.2.1.2. Collision PDU

Position paper 92-30, "DIS Position Paper on Changes to the Collision PDU", was presented to the 7th workshop and discussed the current collision PDU's inability to allow for the conservation of momentum. The recommendation of the paper was to use four of the bits which are currently padding as follows:

Bit 0 (LSB) and 1 identifies how the simulator generating the collision computes collisions internally. The following values apply:

0 - Elastic

1 - Inelastic

2 - Other

Bit 2 and 3 (MSB)identifies which simulator should compute the resulting collision dynamics. The following values apply:

0 - Don't Care

1 - Sending simulator computes collision dynamics

2 - Receiving simulator computes collision dynamics

3 - Each simulator computes their own collision dynamics

This structure will allow the arbitration of control between two vehicles in a collision. However to compute the collision the simulators will need additional data such as the quantity of energy lost and the location of the energy loss.

The ITMC working group did not vote on this recommendation.

5.1.2.1.3. Simulation Management

The 7th workshop reviewed and updated the Strawman Simulation Management protocol.

Changes discussed during the 7th Workshop. The proposal to allow a change in the ratio of simulated time to real-time, both slower and faster was discussed. It was decided that this could be accommodated with the existing Action Request PDU. The need to define an additional 32-bit field for calendar time was discussed. This is necessary for initializing simulation time to Greenwich mean time. The subgroup decided to note in the standard that the Event PDU is asynchronous.

The subgroup decided that it needed more information from the network management subgroup of the Communication Architecture and Security Subgroup (CASS) on the process and procedure for communicating network addresses to the simulation manager.

The FECFR requirements for variations in the Stop Freeze PDU were discussed.

Two unique values for the Entity ID were reserved for simulation management. The Site ID of all ones, means that the PDU is being sent to every entity in the

exercise. All ones in the Host ID means that the PDU is being sent to all entities in the site, and all ones in the Entity ID means that the PDU is being sent to all entities on the Host. Similarly, all zeros in the Site ID means that no entity within the exercise is required to process the PDU. The same is true with Host and Entity.

Another addition to the standard is that management entities will have a unique ID. That is, an entities ID can't be the same and entity ID of the simulation manager.

5.1.2.1.4. Emissions

During the 7th workshop the Emissions Subgroup made several recommendations for the modification of the Emission PDU in Version 2.0. These are summarized in the Emission Subgroup minutes [IST, 1992b] Volume I pp. 113-115 and Volume II p. 110.

5.1.2.1.5. Radio Communications

The Radio Communications Subgroup reviewed the first draft that was included in Version 2.0. There recommended changes are summarized in the Radio Communications Subgroup minutes [IST, 1992b] Volume I pp. 117-120 and Volume II pp. 111-113..

5.1.2.2. Discussion of Potential additions to the DIS Protocol

5.1.2.2.1. Dynamic Terrain PDU/Protocol

The Simulated Environment: Land Subgroup is working on developing recommendations for a Dynamic Terrain Protocol to communicate construction of berms, craters and ditches. The goal is to have an iron-man of the PDU ready to submit to the communications group by the next workshop.

The wood-man of this PDU is currently under development at IST.

5.1.2.2.2. Environment PDU

One of the further goals of the Simulated Environment: Land Subgroup is to formalize requirements for the environment server.

5.1.2.2.3. Newtonian Protocol

During the ITMC subgroup meeting, an interim report on the Newtonian Protocol as DIS protocol extension for logistic simulation was presented. This paper appears in Volume II of the 7th Workshop's Summary Report [IST, 1992b] p. 145-159. This effort was recognized as having promise not only for logistics

but also for collision resolution and perhaps the Mount/Dismount Infantry problem.

5.1.2.2.4. Mount/Dismount PDU

At the 7th workshop a paper was submitted by IST to propose a Mount and Dismount PDU. These PDUs would bring the mounting and dismounting of infantry clearly within the scope of the DIS protocol. After some discussion, it was decided that this functionality may be within the scope of the Newtonian Protocol that is described earlier.

5.1.2.2.5. Concatenated PDU

The ITMC Subgroup is still considering a Concatenation PDU that could be used to reduce network traffic problems.

5.1.2.2.6. Aggregate Protocol Extension

The Aggregate Protocol was first discussed at the 6th workshop. The 7th workshop proposed several changes to this protocol. These changes are summarized in [IST, 1992b] Volume I p. 112.

5.1.2.2.7. Assume Control Protocol

The requirement for an Assume Control Protocol was identified during the 7th workshop. There are various applications for this. First for simulation application hand-over of mine fields and sonobuoys, or munitions hand-over to the target entities. Additional uses for this protocol that were discussed include, handing-over of smoke clouds from tanks or aircraft to an environment server or simulation, hand-over of burning tanks that would allow maintenance of the carcass of the tank on the battlefield while the manned simulation was reconstituted.

5.1.3. Summary of Changes made for the December '92 IEEE Reballot

Version 1.0 of the standard was balloted in April 1992, see Figure 1. In order to become a IEEE standard 75% of all balloters must vote for approval. On the initial ballots the standard received 72% approval and 384 comments. The DIS Steering Committee formed a tiger team to respond to the comments and codified those responses into one single package. Several issues in this package were submitted to the Interface Time Mission Critical (ITMC) working group for a vote at the 7th workshop. This package was submitted back to the IEEE working group which will submit those responses back to the balloters who then had an opportunity to change there vote in a reballot (Dec. 1992).

The following is a summary of the types of comments received after the April 1992 IEEE ballot.

• Change Title	28
• Change Scope	31
• Add Clarification	115
• Correct Error	37
• Editorial	195
• Delete Annexes/Appendices	22
• Change Annexes/Appendices	33

The DIS Steering Committee Tiger Team responded to the comments and made minor adjustments to the standard, see 5.1.3.1. Several substantial changes were recommended and these changes were referred to the ITMC working group with a recommendation for approval. These issues are discussed in section 5.1.3.2.

5.1.3.1. Minor changes made for the IEEE reballot

Change Title: Many comments related to the title change that came about because IEEE changed the title of the standard and many people thought the new title was inappropriate. The title for the standard for the reballot will be: *Standard for Information Technology - Protocols for Distributed Interactive Simulation Applications*.

Classifications and Errors: Many of the requests for clarification and editorial corrections were made to the standard for the reballot.

Annexes/Appendices: Since the Annexes/Appendices will change a lot and the standard shouldn't have to be updated when something changes in the annexes, many of the comments recommended that they not be included in the standard. The Annexes have been removed from the standard and are currently being maintained by IST. As a long term solution to the maintenance of the annexes, an agreement has been made with Defense Information Systems Agency (DISA) to take configuration control of the annexes.

5.1.3.2. ITMC working group recommendations.

The ITMC working group made the following decisions at the 7th workshop on the questions and issues that were referred to it by the steering committee.

- Change BAMs to Radian? The angular representation in the IEEE standard will be radians.

- Change the parameters in the Articulated Part Record to be floating point variables? This passed unanimously.

- Change the representation of time? First, ITMC decided to break the issue into a decision on whether to reference time from the current hour or time from a fixed starting point. It was decided that *time will be referenced from the current hour*. Another issue was whether to change the time scale. The least significant bit of the time field is currently 1.676 microsecon[illegible] Some people felt that another value was needed for a least significant bit of [illegible]at time field. The options of

microseconds and milliseconds were discussed. A vote was taken on whether to change the time scale from 1.676 microseconds (which rolls over on an hourly basis) or to microseconds (which rolls over in about a 45 1/2 minute cycle). *The decision was to stay with the original version and let the clock roll over on the hourly basis.*

• Put Length of the PDU in the PDU Header? The last issue brought before the ITMC group was whether to have the length of a PDU in the PDU header. This would aid in processing concatenated PDUs. *The decision of the group was to use the fourth field of the PDU header (which was padding) to contain the length of the PDU in four byte words.* The question of PDUs that exceed the maximum length was left for further discussion.

5.2. 13TH I/ITSEC LESSONS LEARNED

The Distributed Interactive Simulation Interoperability Demonstration at the 13th I/ITSEC in December 1992 was the fiıst large scale implementation of the DIS protocol.

5.3. ALSP

The purpose of the Aggregate Level Simulation Protocol (ALSP) is to network existing high level simulations (e.g., wargames) for purposes of education and training. ALSP is being developed similarly to SIMNET in that there is no central node and changes in public attributes and external events are broadcast on the network. ALSP operates at a generally higher level of aggregation than DIS. It is possible that eventually the DIS protocols will grow to accommodate ALSP and vice versa. The ADST Architecture and Standards effort will continue to monitor ALSP for developments in this area.

5.4. J-MASS

The Joint Modeling and Simulation System (J-MASS) program has an initial charter to develop a highly detailed non real-time emulation of the SA-12 system operating in an electronic warfare and natural environment as well as to provide modeling and simulation libraries and data dictionaries to support this effort. J-MASS operates at a generally much higher level of detail than does DIS. It is possible that eventually the DIS protocols will grow to accommodate J-MASS, however, or share some commonality in the database or PDU approach. The ADST Architecture and Standards effort will continue to monitor J-MASS for developments in this area.

APPENDIX A:
PROTOCOL EXTENSION TEMPLATE

Appendix A defines a template for documenting protocol extensions.

1. [PROTOCOL EXTENSION]

This section will briefly describe the application protocol extension. A protocol extension may be new sub-protocol, a group of PDUs and instructions that augments and existing protocol/sub-protocol, a single PDU and instructions, or updates and changes to an existing protocol/sub-protocol. If the protocol extension is for new sub-protocol or a group of PDUs the information in sections 1 and 2 may be more extensive.

1.1. BASE STANDARD

This section shall summarize what standard this protocol extension is building on i.e. IEEE 1278-1993, SIMNET 6.6.1, DIS 2.0 (March 1993). This paragraph shall include, by reference, any other protocol extensions that are assumed. Version information, when available, should be included. Reference standards should be included in the applicable documents section 1.2.

This section will highlight any impact that this protocol extension has on the base standard. Any information that should be added/changed in the base protocol should be discussed. Major dependencies with the base standard should also be noted.

This section will also address whether this protocol extension should be integrated into the Base Standard. Test specific extensions may not be integrated into the base standard. Compatibility and integration issues with other DIS standards efforts should be addressed, i. e. Communication Architecture for Distributed Interactive Simulation (CADIS).

1.2. APPLICABLE DOCUMENTS.

The following documents are referenced in this protocol extension:

1.3. IMPLEMENTATION HISTORY.

This section will summarize the environments in which this protocol extension has been implemented.

2. GENERAL REQUIREMENTS

The material for in this section should be suitable for insertion in Section 4 "General Requirements" of the DIS protocol standard [IEEE 1993]. If this is a SIMNET protocol extension, the information should still be in this format.

2.1. INTRODUCTION

This section contains requirements concerning the content and use of this protocol extension in exercises.

2.1.1. Terminology

Define any new terms added by this protocol extension.

2.1.2. Key Concepts

Describe any new key concepts added by this protocol extension.

2.1.3. Information common to all PDUs in this extension. (optional)

2.1.4. General information on Protocol Extension (optional)

2.2. PDUs FOR [PROTOCOL EXTENSION]

The following paragraphs shall establish the content and the procedure for use of the PDU(s) in this protocol extension

2.2.x. [PDU name] for each PDU in extension

This paragraph shall summarize the purpose of the PDU. For example "The Fire PDU shall be used to communicate information associated with the firing of a weapon."

2.2.x.1. Information Contained in the PDU (required)

This paragraph will summarize the type of information contained in the PDU.

2.2.x.2. Issuance of the [PDU Name] (required)

This paragraph defines the conditions under which the PDU should be issued. This paragraph defines the requirements placed upon the issuer of the PDU by the protocol extension. For example "The Fire PDU shall be issued by an entity at the moment it fires a weapon."

This paragraph should also define the quality of communication service that is required by this PDU. For example "The Detonation PDU shall be issued using a real-time, best effort, multicast communication service."

2.2.x.3. Receipt of the [PDU Name] (required)

This paragraph defined the how the PDU shall be interpreted upon receipt. This paragraph defines the requirements placed upon the receiver of the PDU by the protocol extension.

2.2.x.4. Special information related to [PDU Name] (optional)

This optional paragraph may be used to clarify the interpretation of the PDU or special conditions that may need additional explanation.

2.2.x.5. Examples of extension (optional)

This optional paragraph will provide examples of how this PDU is used.

2.2.2. Examples of protocol extension

This optional paragraph will provide examples of how this protocol extension is used. This paragraph should include examples of where multiple PDUs are used in concert to perform composite functions. The paragraph may have a subparagraph for each composite function that is explained. For example, see Figure 1.

4.4.6.5.1.3 Entity Creation. In the case that necessary entity data and initialization data have already been established off-line and prior to the exercise, it is possible to create a new entity by assigning a particular entity ID. To create a new entity, the Simulation Manager, SM, shall issue a Create Entity PDU to the simulation application that will be controlling the simulation entity. The receiving simulation application shall respond with an Acknowledge PDU.

The process of entity creation is illustrated in Figure 4-5.3.

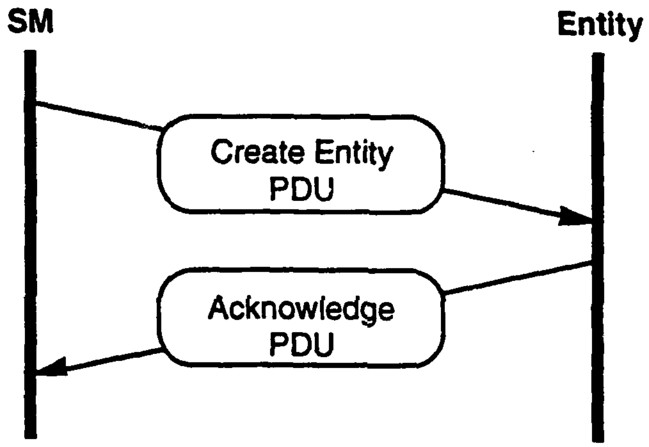


Figure 5-4.3 Entity Creation

Figure 1.. Protocol Example.

3. DETAILED REQUIREMENTS

The material for in this section should be suitable for insertion in Section 5 "Detailed Requirements" of the DIS protocol standard. If this is a SIMNET protocol extension, the information should still be in this format.

3.1. INTRODUCTION

This section defines the PDUs and their fields.

3.2. REPRESENTATION OF DATA

This paragraph will note any variation with the base standard's representation of data.

3.3. BASIC DATA TYPES AND RECORDS

This paragraph will define any basic data items or records that are added/changed by this protocol extension. Data items and records should be included in this section if there are multiple references to them or they are useful outside of this protocol extension.

3.4. LIST OF PDUS IN PROTOCOL EXTENSION

This paragraph will list the PDUs that comprise this protocol extension.

3.5. PROTOCOL DATA UNITS FOR PROTOCOL EXTENSIONS

3.5.x. PDU information (for each in protocol extension)

This paragraph shall briefly summarize the PDU and then give a detailed description on each of the fields in the PDU. For example see Figure 2.

```
The firing of a weapon shall be communicated by issuing a
Fire PDU.  The Fire PDU shall contain the following fields:

     (1)        PDU Header - These fields shall identify
           the protocol version number, the exercise
           identifier, the protocol data unit type and the
           length of the PDU.  The PDU Header shall be
           represented by the PDU Header Record (see
           5.3.15).

     (2)        Firing Entity Identification - This field
           shall identify the firing entity.  This field
           shall be  represented by an Entity Identifier
           Record (see 5.3.8).

     ...

     (10)  Range - This field shall specify the range that
           an entity's fire control system has assumed in
           computing the fire control solution.  This field
           shall be represented by a 32-bit floating point
           number in meters.  For systems where range is
           unknown or unavailable, this field shall contain
           a value of zero.
```

Figure 2. Example PDU Description.

This paragraph shall also represent the PDU as a figure. For example "The Repair Complete PDU is represented in Figure 3.

Record Name or Field Name

Field Size (bits)	REPAIR COMPLETE PDU FIELDS	
32	PROTOCOL HEADER	Protocol Version - 8 bit unsigned integer Exercise ID - 8 bit unsigned integer PDU Type - 8 bit enumeration Length - 8 bitunsigned integer
48	RECEIVING ENTITY ID	Site - 16 bit unsigned integer Host - 16 bit unsigned integer Entity - 16 bit unsigned integer
48	REPAIRING ENTITY ID	Site - 16 bit unsigned integer Host - 16 bit unsigned integer Entity - 16 bit unsigned integer
16	REPAIR	16 bit enumeration
16	PADDING	16 bit unused

Field Name and Length or Field Length

Figure 3. Example PDU Diagram.

4. ENUMERATED AND BIT ENCODED VALUES FOR USE WITH (PROTOCOL EXTENSION)

The material for in this section should be suitable for insertion in the "Enumerated and Bit Encoded Values for Use with Protocols for Distributed Interactive Simulation Applications" document that accompanies the base standard (see section 1.1). If this is a SIMNET protocol extension, the information should still be in this format.

4.1. UPDATED FIELDS

For each enumerated or bit encoded field that is changed by this protocol extension the following information will be provided.

- field name and description
- field value and meaning
- impact on other simulation application

4.2. NEW FIELDS

New fields will provide the following information in addition to the information for updated fields.,

- Scope
- Applicable Documents

APPENDIX B:
DIS PROTOCOL EXTENSIONS

Component Protocol Extensions

B1 C3I Protocol Extension

B2 Digital Message Communications Protocol Extension

APPENDIX B1: C3I PROTOCOL EXTENSION

TABLE OF CONTENTS

1. C3I PROTOCOL EXTENSION

The C3I Protocol Extension has been developed to enable cognitive information to communicated in the DIS environment.

1.1. BASE STANDARD

This is a protocol extension to the DIS 2.0 Standard [IST 1993a].

1.2. APPLICABLE DOCUMENTS.

IEEE, 1993	1278-1993, Standard for Information Technology, Protocols for Distributed Interactive Simulation Applications, IEEE, New York, NY, March 1993
IST, 1993a	Proposed IEEE Standard Draft: Standard for Information Technology - Protocols for Distributed Interactive Simulation Applications, Version 2.0 - Second Draft, IST-CR-93-01, UCF Institute for Simulation and Training, 22 March 1993.
IST, 1993b	Enumeration and Bit Encoded Values for Use with Protocols for Distributed Interactive Simulation Applications, Version 2.0 - Second Draft, IST-CR-93-02, UCF Institute for Simulation and Training, 22 March 1993.

2. GENERAL REQUIREMENTS

TBS

2.1. INTRODUCTION

TBS

2.1.1. Terminology

TBS

2.1.2. Key Concepts

TBS

2.1.3. General information on C3I protocol

PDUs FOR INITIALIZATION:
- Correlation
- Perception Control
- Non-Point Control Measures
- Point Control Measures
- Point Control Measure with Relations

PDUs FOR COMMUNICATIONS:

Entity Communication
Ctrl Measures for Commo
Entity Locations
Task Organization
General Purpose Request
Perceived Status
Perceived Tactics
Incident / Situation

PDUs FOR SIMULATION CONTROL:

(Re)Start
Admin Request
Impending Admin Action

2.1.4. Information common to all PDUs in C3I protocol

2.2. PDUs FOR C3I PROTOCOL EXTENSION FOR CGF

The following paragraphs shall establish the content and the procedure for use of the PDU(s) in this protocol extension

2.2.1. Incident/Situation PDU

PDU for announcing that in incident has occurred

2.2.1.1. Information Contained in the Incident PDU

See the detailed description of this PDU in section 3.4.

2.2.1.2. Issuance of the Incident PDU

This PDU will be issued anytime an incident has occurred.

2.2.1.3. Receipt of the Incident PDU

Receiving entities may take application specific action upon receipt of Incident PDUs.

2.2.2. Correlation PDU

This format is used to relate a code to a character string and to define any supporting information. Examples of uses are to relate a code with a contingency plan and define parameters to the contingency plan; or to relate a code with a situation. This is an update to the previously defined Correlation PDU.

In cases where supporting information is defined for a Correlation Code, a separate PDU is formed for the initial correlation (with 0 for SupportingInfoOrder), and formed for each piece of supporting information (with a positive integer is assigned to the SupportingInfoOrder to define the order of supporting information). Each PDU defined for supporting correlation information will have the same Correlation Site, Host and Code as the correlation being supported.

2.2.2.1. Information Contained in the Correlation PDU

See the detailed description of this PDU in section 3.4.

2.2.2.2. Issuance of the Correlation PDU

This PDU is typically issued at the start of an exercise.

2.2.2.3. Receipt of the Correlation PDU

Upon receipt a table should be constructed to related correlation codes received in PDUs to the corresponding character strings.

2.2.2.4. Special information related to Correlation PDU

For the June demonstration, the SupportingInfoOrder will always be 0. The effect of this is that only correlation codes will be defined, not supporting info.

2.2.3. Perception Control PDU

This PDU will be used to control perception reports for entities.

2.2.3.1. Information Contained in the Perception Control PDU

See the detailed description of this PDU in section 3.4.

2.2.3.2. Issuance of the Perception Control PDU

TBS

2.2.3.3. Receipt of the Perception Control PDU

TBS

2.2.3.4. Special information related to (Re)Start PDU

For the June Demo:

For each perception belonging to an entity, an Entity State PDU will be sent instead of the Entity Location PDU. Since we will use dedicated ports for monitor and control, there will be no confusion about the perception owner. In addition, the following conventions will be used:

- use zeros for unknown values.
- no shapes
- add bit flag to say dropped (in EntityAppearance, same one used for ATACMS/MLRS in JayHawk Thunder)
- for smoke cloud, use Entity Type Record value 41001000
- Entity Site and Host will be meaningless. Entity ID will really be the Track Number for the aggregated perception.
- EntityMarking will contain the first eleven characters of the organization name.

2.2.3.5. Example of the Perception Control PDU

A Perception Control PDU will be used to request perceptions for an entity. Perceptions for an entity will be started with an Entity Communication PDU to associate an ID with the perception report. Each perception will then be reported with a Entity Location PDU.

2.2.4. Entity Communication PDU

This PDU will be used to describe communication between entities. Examples of uses are Perceptions, OPORD, FRAGO, Warning Order, SITREP, OPREP, etc.

The Communication information defined in this PDU will be used to relate information reported in subsequent PDUs to information reported here.

The Related Commo Site, Host and ID can be used to relate a FRAGO to the OPORD which it is modifying. If a relation to a previous Communication is not necessary, these fields will contain zeros.

The Correlation Site, Host, and Code can be used to identify the type of communication which this PDU is announcing. These fields are set up (typically) at initialization by the CGF.

2.2.4.1. Information Contained in the Entity Communication PDU

See the detailed description of this PDU in section 3.4.

2.2.4.2. Issuance of the Entity Communication PDU

TBS

2.2.4.3. Receipt of the Entity Communication PDU

TBS

2.2.5. Task Organization PDU

This PDU will be used to describe the subordinates of a single entity. It can also define the order of succession.

2.2.5.1. Information Contained in the Task Organization PDU

The Communication information will be the same on any task organization to be used for one OPORD, one FRAGO, or one Warning Order. For initialization purposes, Commo information may be 0. The CommoPartID is used to distinguish between PDUs for this Communication.

The Entity identification is used to identify the unit whose subordinate units are listed.

A simple method of succession of the subordinates is accomplished by the order in which the subordinate units are reported in the PDU. A more complicated succession may be defined in the tactics and contingency plans of the Model Input for the Entity. If an additional method of succession is to be defined, an Association PDU may be used (using the CommoPartID) to attach a succession plan to the Entity. This allows definition of a more specific set of rules for determining when succession should occur and how. The succession plan is identified in the Correlation PDU at initialization.

The time stamp represents an effective time for the task organization. (The current time stamp is insufficient for future and past since it only can show 1 hour's worth of time. Because of this drawback, we will now be using a 64 bit time stamp which

reflects Game Time in all PDUs. A new PDU will be designed to transmit the starting game time as part of initialization.)

The Subordinate Entities are defined as a variable list at the end of the PDU. The number of entries in the list can be found in NumSubordinates.

Command chains are at a player level not at a platform level, therefore the track number reported is the Monitor Unit's aggregation of the platforms into a player. This equates to calling the S1, S2, S3, S4 and S5 a company headquarters. To the CGF Engine, the company HQ is a player with a platform for each of the HQ elements. It does not make sense to report the HQ elements in the command chain, but it does make sense to report the company HQ.

The OrgName and Type Code for a unit will be reported in the Notion or Perception PDU(s) for that unit. Any unit in a player's Task Org/Command Chain is also on his perception list.

See the detailed description of this PDU in section 3.4.

2.2.5.2. Issuance of the Task Organization

TBS

2.2.5.3. Receipt of the Task Organization

TBS

2.2.5.4. Examples of the Task Organization PDU

OPORD/FragO Examples

Since a FragO is a subset of an OPORD, the example given applies equally to both.

Entering the Order through the UI:

Correlation PDUs set up any menu options for tactics, types of things, contingency plans, etc. The Correlation PDUs are typically sent at the start of a simulation, but not necessarily.

Sending the Order to the CGF:

To report an OPORD or FRAGO, send an Entity Communication PDU which sets up the identifying numbers for the communication as well as the sender and recipient(s). The identifying numbers are used in subsequent PDUs to identify information belonging to the communication.

For each part of the OPORD, a short description of the PDUs expected to transmit the information will be given.

OPORD/FRAGO # - The Commo Site, Host, and ID distinctly identify this order. The U.I. can have a mapping of that unique ID to some user entered ID if desired.

Reference - No map reference is needed, the map(s) being used are set up in the input files and should be the same across CGF and U.I. A reference can be made to a previous communication through the RelatedCommo Site, Host, and ID of the Entity Communication PDU.

Time Zone for Order - This value should never be needed as the times entered by the user should always be converted to Game Time when entered in the PDUs.

Task Organization - Task Organization changes should be reported via multiple Task Organization PDUs.

Situation - All information contained in the friendly or enemy situation can be conveyed through multiple Ctrl Measure PDUs. (It is our contention that anything about the current situation can be graphically displayed, and therefore can be reported as Ctrl Measures.) Assumptions are not needed for the CGF. Attachments and Detachments can be reported similarly to Task Organization.

Mission - and

Execution - Most of the mission and execution can be shown as control measures graphically and can therefore be reported via multiple Ctrl Measure PDUs. Information about time to reach objectives can also be reported with the objective reported in the Ctrl Measure PDU. Information like "Don't get decisively engaged" or "Withdraw upon contact" can be reported as contingency plans attached to a unit.

Service Support - The information reported in Service Support can be reported in the Task Organization or as Friendly Situation.

Command and Signal - Most command and signal information can be reported as Friendly Situation. Occasionally, a contingency plan may need to be attached to a unit or units to force a certain type of behavior (listening silence, complex succession of command, etc.). Simple succession of command may be reported in the same PDUs as the Task Organization.

When all PDUs for a communication have been sent, send an Entity Communication Termination PDU to let the CGF know that it has received the complete set of PDUs for the communication. This PDU summarizes the number of PDUs of each type sent for this communication.

2.2.6. (Re)Start PDU

This PDU will be issued just prior to starting the simulation or restarting the simulation (after being paused). The ScenarioTimeUnits field is used to specify the units of measure for the time reported as ScenarioTime. For most uses, the time units will be according to the Gregorian Calendar.

TimeCoordinate X, Y, and Z define the point on the earth from which all Scenario times are valid. This prevents a problem with time zones, in fact, the time should not even be time zonal, but sidereal.

The GameStartTime is typically 0 at simulation start, but can be any number. The game time is used to communicate times throughout the execution of the simulation.

CountToStart is reports the delta amount of time from receipt of this message until the simulation resumes/starts execution. (There is always a difference between the time each asset will have - that is transmission time. No fix is proposed for this.)

The RealTimeMult is the fraction of wall clock time at which the simulation will proceed.

2.2.6.1. Information Contained in the (Re)Start PDU

See the detailed description of this PDU in section 3.4.

2.2.6.2. Issuance of the (Re)Start PDU

This PDU will be issued just prior to starting the simulation or restarting the simulation (after being paused).

2.2.6.3. Receipt of the (Re)Start PDU

TBS

2.2.6.4. Special information related to (Re)Start PDU

For the June Demo:

ScenarioTimeUnits will be 1.

ScenarioStartTime will contain the number of seconds since midnight (GMT), January 1, 1970.

TimeCoordinate X, Y, and Z will reflect a point in the scenario playbox.

GameStartTime will begin at 0, for subsequent Admin requests, will reflect current game time.

CountToStart is typically 5 seconds.

RealTimeMult will typically be 1.0, but can be changed via Admin Requests.

2.2.7. Admin Request PDU

This PDU will be sent to the Master Controller (CGF Engine) whenever changes to the execution parameters of the simulation are desired.

2.2.7.1. Information Contained in the Admin Request PDU

The ActionRequested field specifies the administrative action to be taken. Option 1 will cause a pause in the simulation, Option 2 will cause the real time multiple to be modified. Option 3 will cause the simulation to resume operation after being frozen. Option 4 combines options 2 and 3, allowing a resume of operations with a changed real time multiple.

The EffectiveGameTime is the requested effective time of the administrative change.

CountToStart is the number of wall clock seconds until the change should take effect.

EffectiveGameTime and/or CountToStart can contain -1 to signify as soon as possible. For Option 1 or Option 2, EffectiveGameTime is the time at which the request should take effect; CountToStart could be used instead. If EffectiveGameTime is used for Option 1 or Option 2, CountToStart is ignored.

For Option 3 or Option 4, EffectiveGameTime specifies the game time to start at (this could cause a jump in the game). A value of -1 causes the game to resume at the point where it was paused. Regardless of the EffectiveGameTime, CountToStart gives a delta time to restart the game. A value of -1 for CountToStart means restart when ready.

The RealTimeMult is the fraction of wall clock time at which the simulation will proceed. If option 2 or 4 are chosen, this value will modify the real time multiple for the simulation.

See the detailed description of this PDU in section 3.4.

2.2.7.2. Issuance of the Admin Request PDU

TBS

2.2.7.3. Receipt of the Admin Request PDU

TBS

2.2.7.4. Special information related to Admin Request PDU

For the June Demo:

All options will be available. This capability should be restricted to a controller station (Ground Truth?).

Use -1 for EffectiveGameTime and CountToStart for any requests to keep things simple.

2.2.8. Impending Admin Action PDU

This PDU will be issued by the Master Contr ller (CGF Engine) whenever changes to the execution parameters of the simulation are going into effect.

The ActionRequested field specifies the impending administrative action. Option 1 will cause a pause in the simulation, Option 2 will cause the real time multiple to be modified.

The EffectiveGameTime is the effective time of the administrative change. If option 2 is chosen, this value represents the new real time multiple for the simulation.

2.2.8.1. Information Contained in the Impending Admin Action PDU

See the detailed description of this PDU in section 3.4.

2.2.8.2. Issuance of the Impending Admin Action PDU

TBS

2.2.8.3. Receipt of the Impending Admin Action PDU

TBS

2.2.9. General Purpose Request PDU

This PDU will replace the Service Request PDU since the ServiceRequest is limited to resupply and repair, and time and place for delivery/link up is not specifiable.

2.2.9.1. Information Contained in the General Purpose Request PDU

See the detailed description of this PDU in section 3.4.

2.2.9.2. Issuance of the General Purpose Request PDU

TBS

2.2.9.3. Receipt of the General Purpose Request PDU

TBS

2.2.9.4. Special information related to General Purpose Request PDU

For the June Demo:

All options will be possible.

2.2.10. Perceived Status PDU

2.2.10.1. Information Contained in the Perceived Status PDU

See the detailed description of this PDU in section 3.4.

2.2.10.2. Issuance of the Perceived Status PDU

TBS

2.2.10.3. Receipt of the Perceived Status PDU

TBS

2.2.11. Entity Locations PDU

This PDU will be used to describe entity locations, etc.

2.2.11.1. Information Contained in the Entity Locations PDU

The PercepFlag is used to define general characteristics of the Perception.

The Entity identification is used to identify a unit whose location is reported (proposed or actual, perceived or real location); to identify a unit who is associated with a control measure (i.e., Security Operations).

EntityAppearance contains the perceived appearance of a unit.

EntityType distinguishes among units.

Correlation Site, Host, and Code are valid only if PercepFlag C is 1 (i.e. org name is known).

The location of the perception is specified.

The time stamp represents an effective time for the control.

See the detailed description of this PDU in section 3.4.

2.2.11.2. Issuance of the Entity Locations PDU

TBS

2.2.11.3. Receipt of the Entity Locations PDU

TBS

2.2.11.4. Special information related to Entity Locations PDU

For the June Demo:

Subsequent points for lines or areas will be absolute (DataFlag B will equal 1).

2.2.12. Point Control Measures PDU

This PDU will be used to describe single point control measures.

A number of point control measures may be defined in one Point Control Measures PDU. The NumCtrlMeasures identifies the number of separate control measures reported. Any control measure that is defined only by a point will be reported in this PDU.

2.2.12.1. Information Contained in the Point Control Measures PDU

The CtrlMeas Site, Host, and ID are global ids which are used outside the scope of any single Entity Communication.

A tactic may be associated with the control measure through the Correlation Site, Host, and ID.

EntityType distinguishes among control measures.

The location or origin of the Control Measure is specified.

The time stamp represents an effective time for the control.

See the detailed description of this PDU in section 3.4.

2.2.12.2. Issuance of the Point Control Measures PDU

TBS

2.2.12.3. Receipt of the Point Control Measures PDU

TBS

2.2.12.4. Special information related to Point Control Measures PDU

For the June Demo:

This PDU will be used to define simple control measures.

Contact Point - Point Control Measure with Relations PDU will be used.

Control/Coordination Point - same as for Contact Point

Passage Point - Point Control Measure PDU will be used. The Correlation Site, Host and Code are used to define a tactic associated with the passage point. The Passage Point may also be defined as a line, area or volume, in which case the Non-Point Control Measure PDU must be used.

Release Point - Location may be explicitly defined in a Point Control Measure PDU or may be defined as part of the route which it delineates. If it is defined explicitly, a Point Control Measure with Relations PDU must correlate it with the route.

Start Point - same as for Release Point.

Target Location - Location is defined in the Entity Location PDU. The time stamp is used to differentiate between past, present, and future locations.

Reference Point - Location is defined in the Point Ctrl Measures PDU. Altitude may be used.

Unit Location - same as for Target Location.

2.2.13. Point Control Measures with Relations PDU

This PDU will be used to describe a point control measure and to relate it to other previously defined control measures. (This can be used to define a Coordination Point location and relate it to the Phase Line and Lateral Boundary which cross at the point.)

One point control measure may be defined in one Point Control Measure with Relations PDU. In addition, multiple previously defined control measures may be related to the point control measure defined. The CtrlMeas Site, Host, and ID are global ids which are used outside the scope of any single Entity Communication.

A tactic may be associated with the control measure through the Correlation Site, Host, and ID.

2.2.13.1. Information Contained in the Point Control Measures with Relations PDU

EntityType distinguishes among control measures.

The location or origin of the Control Measure is specified.

The time stamp represents an effective time for the control.

See the detailed description of this PDU in section 3.4.

2.2.13.2. Issuance of the Point Control Measures with Relations PDU

TBS

2.2.13.3. Receipt of the Point Control Measures with Relations PDU

TBS

2.2.13.4. Special information related to Point Control Measures with Relations PDU

For the June Demo:

This PDU will be used to define simple control measures.

2.2.14. Non-Point Control Measures PDU

This PDU will be used to describe control measures consisting of multiple points.

One non-point control measure may be defined in one Non-Point Control Measures PDU. The CtrlMeas Site, Host, and ID are global ids which are used outside the scope of any single Entity Communication.

2.2.14.1. Information Contained in the Non-Point Control Measures PDU

A tactic may be associated with the control measure through the Correlation Site, Host, and ID.

EntityType distinguishes among control measures.

The location or origin of the Control Measure is specified.

The time stamp represents an effective time for the control.

The additional points needed to represent a line or area are given in the repeating field.

See the detailed description of this PDU in section 3.4.

2.2.14.2. Issuance of the Non-Point Control Measures PDU

TBS

2.2.14.3. Receipt of the Non-Point Control Measures PDU

TBS

2.2.14.4. Special information related to Non-Point Control Measures PDU

For the June Demo:

This PDU will be used to define simple control measures.

Line Control Measures

Phase Line (PL) - Line is defined as a list of points in the Non-Point Ctrl Measure PDU. The Time stamp may be used to correlate multiple unit's arrival at the Phase Line. The Correlation Code may be used to force a certain set of actions related to the line.

Lateral Boundary - Line is defined in the Non-Point Ctrl Measure PDU as a list of points. Altitude may be used. It is not likely that the Time stamp, or Correlation Code are used.

Rear Boundary - same as for Lateral Boundary.

Fire Support Coordination Line (FSCL) - same as for Lateral Boundary, except that the Correlation Code is used to identify the tactic associated with the measure, and time stamp is probably used to identify when the measure goes into effect.

Line Of Attack (LOA) - same as for FSCL

Line of Departure (LD) - same as for FSCL

Line of Contact (LC) - same as for FSCL

Line of Departure is Line of Contact (LD/LC) - same as for FSCL

Sector - The Unit with Related Ctrl Measures PDU is used to identify the unit and relate the control measures defining its sector of operations with it.

Security Operations - Same as for Sector, except Correlation Code is used to identify which type of security operations is to be used.

Area Control Measures

Assembly Area - Area is defined in the Non-Point Ctrl Measure PDU. A tactic may be attached to this area through the Correlation Code. A time stamp is likely to be used. A Unit with Related Ctrl Measures PDU should be used to relate entities to the Assembly Area.

Objective - same as for Assembly Area.

No Fire Area - same as for Assembly Area, except a tactic of No-Fire should be defined in the Correlation Code.

Obstacles - Area is defined in the Non-Point Ctrl Measure PDU. A Time stamp may be used. The Entity Type Record should identify the type of obstacle. A Unit with Related Ctrl Measures PDU could relate entities with the obstacle.

2.2.15. Ctrl Measures for Commo PDU

This format is used define which previously defined control measures should be used with this communication.

2.2.15.1. Information Contained in the Control Measures for Commo PDU

See the detailed description of this PDU in section 3.4.

2.2.15.2. Issuance of the Control Measures for Commo PDU

TBS

2.2.15.3. Receipt of the Control Measures for Commo PDU

TBS

2.3. EXAMPLES OF C3I PROTOCOL EXTENSION FOR CGF

Instructions for SITREP creation:

The PDUs expected in a SITREP are listed below. For each, the fields of the PDU are defined with reference to SITREPs.

Information for "Defensive Measures" (line 6) will not be supplied by the CGF Engine. It can be left in the format for probable future use, or discarded.

Only Tactics are available for "Summary of Unit Activity" (line 3), and "Enemy Activity/Intentions" (line 7) for the June Demo.

The information received in the Perceived Status PDUs can be used to formulate a FLOT if you want to come up with an algorithm; calculate which units would be considered on the Forward Line (based on Loss and Gain Rates); plot the lines, etc. Our feeling is that there are too many methods of determining the FLOT, and that having it in the SITREP doesn't really gain that much for the viewer...

Entity Commo PDU

Commo Site, Host, and ID - unique to this report

RelatedCommo Site, Host, and ID - probably 0's since SITREPs aren't usually correlated with previous reports.

CommoPartID - incremented from last CommoPartID

fromEntitySite, Host, ID, and TrackNum - contains id for "Unit submitting report" (line 1) with respect to Monitor Unit

CorrelationSite, Host, and Code - contains code for SITREP

TimeOfReport - contains "DTG of Report" (line2)

NumPDUs - contains number of PDUs to be sent for this report

NumRecipients - probably only sent to 1 (superior)

repeat {

toEntitySite, Host, ID, and TrackNum - corresponds to superior unit.

} for each recipient

Perceived Tactics PDU

CommoSite, Host, and ID - set up in Entity Commo PDU

CommoPartID - incremented from last CommoPartID

EntitySite, Host, ID, and TrackNum - contains identification for "Unit submitting report" (line 1) with respect to Monitor Unit

Fidelity - contains 1 for self

NumTactics - contains number of tactics unit submitting report is operating under plus number of tactics unit is intending to use

repeat {

EffectiveTime - contains "DTG of Report" (line 2) or time when tactic is intended to be operational.

CorrelationSite, Host, and Code - identifies a tactic which this unit is operating under or which this unit is intending to follow (distinguishable by Effective Time field). Feeds "Summary of Unit Activity" (line 3) or "Tactical Intentions" (line 8). The tactics would be displayed as: "Tactics for unit a <DTG>: <tactic string>, <tactic string>, ..., <tactic string>.

} for each tactic

Entity Locations PDU

One Entity Locations PDU will list all subordinates by identification and center of mass.

CommoSite, Host, and ID - set up in Entity Commo PDU

CommoPartID - incremented from last CommoPartID

NumEntities - contains number of entities to be reported in this PDU

repeat {

EntitySite, Host, and ID - contains identification for the Monitor Unit.

TrackNum - the number local to the MonitorUnit which is used to represent the entity being reported. This value may be used to distinguish among aggregations of the same entity type (if Flag C, below, is 0). When this number is needed, it is used with entity type for "Unit" (line 4: Location).

PercepFlags - A contains 0 for N/A

B contains 2 for cooperative data

C probably contains 1 for OrgName known

D..I contain 0 for N/A

EntityAppearance - probably all 0s, possibly dead is set

EntityOrgCode Site, Host, and Code - contains semantic code corresponding to org name if Flag C (above) is 1. If valid, this string is used for "Unit" (line 4: Location).

EntityType - corresponding type id from DIS. This entity type is used for "Unit" (line 4: Location).

Timestamp - contains time of last perception update

LocationXYZ - contain center of mass of aggregated unit. Feeds "Center of Mass" (line 4: Location)

} for each subordinate

Perceived Status PDU

All battle resources will be reported with this PDU.

CommoSite, Host, and ID - set up in Entity Commo PDU

CommoPartID - incremented from last CommoPartID

EntitySite, Host, ID, and TrackNum - contains identification for "Unit submitting report" (line 1) with respect to Monitor Unit

Fidelity - contains 1 for self

EffectiveTime - contains "DTG of Report" (line 2)

NumBelongings - contains number of distinct types of resources to be reported

repeat {

EntityTypeRecord - defines type of resources: personnel, equipment, etc.

PresentAmount - amount of this type of resource on hand

TotalLosses - amount of losses sustained to this type of resource

TotalGains - amount of gains for this type of resource

LossRate - num losses per second

GainRate - num gains per second

} for each resource

For each repetition of Resource status, one line under "Battle Resources" (line 5) is filled in.

"Resource" (Column 1) - found by creating a descriptive string for the EntityTypeRecord values reported.

"Status" (Column 2) - found by taking "Authorized" (Column 3) divided by "Operational" (Column 4) and then find the appropriate status indicator (Green, Amber, Red, Black) based on that percentage.

"Authorized" (Column 3) - found by using the following formula on the PDU values associated with the resource type: PresentAmount + TotalLosses - TotalGains

"Operational" (Column 4) - same as PresentAmount reported for the resource type.

A "Summary" status can be found by combining all the "Authorized" (Column 3) values and dividing by the sum of the "Operational" (Column 4) values. The resulting percentage is then referenced to find the appropriate color designator.

Perceived Tactics PDU

CommoSite, Host, and ID - set up in Entity Commo PDU

CommoPartID - incremented from last CommoPartID

EntitySite, Host, ID, and TrackNum - contains identification of enemy unit with respect to Monitor Unit

Fidelity - probably contains 3 for non-cooperative

NumTactics - contains number of tactics enemy unit is believed to be operating under

repeat {

EffectiveTime - contains time of tactics perception

CorrelationSite, Host, and Code - identifies a tactic which the enemy unit is believed to be operating under. Feeds "Enemy Activity/Intentions" (line 7). The tactics would be displayed as: "Tactics for unit at <DTG>: <tactic string>, <tactic string>, ..., <tactic string>.

} for each tactic

<u>Instructions for SPOTREP creation:</u>

The PDUs expected in a SPOTREP are listed below. For each, the fields of the PDU are defined with reference to SPOTREPs.

Only Tactics are available for "Summary of Unit Activity" (line 3), and "Enemy Activity/Intentions" (line 7) for the June Demo.

Entity Commo PDU

Commo Site, Host, and ID - unique to this report

CommoPartID - incremented from last CommoPartID

RelatedCommo Site, Host, and ID - probably 0's since SPOTREPs aren't usually correlated with previous reports.

fromEntitySite, Host, ID, and TrackNum - contains id for "Observer" (line 1) with respect to Monitor Unit

CorrelationSite, Host, and Code - contains code for SPOTREP

TimeOfReport - contains "DTG of Report" (line2)

NumPDUs - contains number of PDUs to be sent for this report

NumRecipients - probably only sent to 1 (superior)

repeat {

toEntitySite, Host, ID, and TrackNum - corresponds to superior unit.

} for each recipient

Entity Locations PDU

A Entity Locations PDU will be used to define the type of unit observed and its location.

CommoSite, Host, and ID - set up in Entity Commo PDU

CommoPartID - incremented from last CommoPartID

NumEntities - probably 1.

repeat {

EntitySite, Host, and ID - contains identification for the Monitor Unit.

TrackNum - the number local to the MonitorUnit which is used to represent the entity being reported. This value may be used to distinguish among aggregations of the same entity type (if Flag C, below, is 0). When this number is needed, it is used with entity type for "Unit" (line 2: What is Observed).

PercepFlags - A contains 0 for N/A

B probably contains 3 for non-cooperative data

C probably contains 2 for OrgName unknown

D..I contain 0 for N/A

EntityAppearance - probably all 0s, possibly dead is set, if so, should be noted in "Activity" (line 2: What is Observed)

EntityOrgCode Site, Host, and Code - contains semantic code corresponding to org name if Flag C (above) is 1. If valid, this string is used for "Unit" (line 2: What is Observed).

EntityType - corresponding type id from DIS. This entity type is used for "Unit" (line 2: What is Observed).

Timestamp - contains time of last perception update. This time feeds "Time" (line 2: What is Observed)

LocationXYZ - contain center of mass of aggregated unit. Feeds "Location" (line 2: What is Observed)

} for NumEntities

Perceived Tactics PDU

CommoSite, Host, and ID - set up in Entity Commo PDU

CommoPartID - incremented from last CommoPartID

EntitySite, Host, ID, and TrackNum - contains identification of entity being reported

Fidelity - probably contains 3 for non-cooperative

NumTactics - contains number of tactics entity being reported is perceived to be operating under

repeat {

EffectiveTime - contains time of last perception update

CorrelationSite, Host, and Code - identifies a tactic which this unit is perceived to be operating under. Feeds "Activity" (line 2: What is Observed). The tactics would be displayed as: "Tactics for unit at <DTG>: <tactic string>, <tactic string>, ..., <tactic string>.

} for each tactic

Perceived Status PDU

All equipment associated with entity being reported will be described with this PDU.

CommoSite, Host, and ID - set up in Entity Commo PDU

CommoPartID - incremented from last CommoPartID

EntitySite, Host, ID, and TrackNum - contains identification for entity being reported with respect to Monitor Unit

Fidelity - probably contains 3 for non-cooperative

EffectiveTime - contains time of last perception update

NumBelongings - contains number of distinct types of resources to be reported

repeat {

EntityTypeRecord - defines type of resources: personnel, equipment, etc.

PresentAmount - amount of this type of resource on hand

TotalLosses - probably 0 for N/A

TotalGains - probably 0 for N/A

LossRate - probably 0 for N/A

GainRate - probably 0 for N/A

} for each resource

For each repetition of Resource status, one phrase of "Equipment" (line 2: What is Observed) is filled in.

Perceived Tactics PDU

CommoSite, Host, and ID - set up in Entity Commo PDU

CommoPartID - incremented from last CommoPartID

EntitySite, Host, ID, and TrackNum - contains identification for "Unit submitting report" (line 1) with respect to Monitor Unit

Fidelity - contains 1 for self

NumTactics - contains number of tactics unit submitting report is intending to use

repeat {

EffectiveTime - contains time when tactic is intended to be operational

CorrelationSite, Host, and Code - identifies a tactic which this unit intends to follow. Feeds "What are actions" (line 3). The tactics would be displayed as: "Tactics for unit at <DTG>: <tactic string>, <tactic string>, ..., <tactic string>.

} for each tactic

<u>Instructions for SHELREP creation:</u>

The PDUs expected in a SHELREP are listed below. For each, the fields of the PDU are defined with reference to SHELREP.

"Azimuth to bursts" (line3) can be calculated from observer location and fire location for display purposes. Or this line can be left blank. The CGF Engine will always report observer location and fire location.

"Flash to bang" (line 10) can be calculated from other information reported, if desired for display. The CGF Engine will not report this value.

Entity Commo PDU

Commo Site, Host, and ID - unique to this report

CommoPartID - incremented from last CommoPartID

RelatedCommo Site, Host, and ID - probably 0's since SHELREPs aren't usually correlated with previous reports.

fromEntitySite, Host, ID, and TrackNum - contains id for "Unit of Origin" (line 1) with respect to Monitor Unit

CorrelationSite, Host, and Code - contains code for SHELREP

TimeOfReport - contains GameTime of start of attack. Feeds "DTG attack started" (line 4).

NumPDUs - contains number of PDUs to be sent for this report

NumRecipients - probably only sent to 1 (superior)

repeat {

toEntitySite, Host, ID, and TrackNum - corresponds to superior unit.

} for each recipient

Entity Locations PDU

One Entity Locations PDU will be used to report the location of the observer and the type and location of the attacker. (The time associated with the attacker is used for end time of attack.)

CommoSite, Host, and ID - set up in Entity Commo PDU

CommoPartID - incremented from last CommoPartID

NumEntities - number entities to be reported - probably 2 (reporter and attacker)

repeat {

(iteration 1: reporter)

EntitySite, Host, and ID - contains identification for the Monitor Unit.

TrackNum - the number local to the MonitorUnit which is used to represent the observer entity. This value may be used to distinguish among aggregations of the same entity type (if Flag C, below, is 0).

PercepFlags - A contains 0 for N/A

B contains 1 for self

C probably contains 1 for OrgName known

D..I contain 0 for N/A

EntityAppearance - probably all 0s, possibly dead is set

EntityOrgCode Site, Host, and Code - contains semantic code corresponding to org name if Flag C (above) is 1.

EntityType - corresponding type id from DIS.

Timestamp - contains time of last perception update

LocationXYZ - contain center of mass of aggregated unit. Feeds "Observer Location" (line 2)

(iteration 2: attacker)

EntitySite, Host, and ID - contains identification for the Monitor Unit.

TrackNum - the number local to the MonitorUnit which is used to represent the attacking entity. This value may be used to distinguish among aggregations of the same entity type (if Flag C, below, is 0).

PercepFlags - A contains 0 for N/A

B contains 3 for non-cooperative data

C probably contains 2 for OrgName unknown

D..I contain 0 for N/A

EntityAppearance - probably all 0s, possibly dead is set

EntityOrgCode Site, Host, and Code - contains semantic code corresponding to org name if Flag C (above) is 1.

EntityType - corresponding type id from DIS.

Timestamp - contains end time of attack, Feeds "DTG attack ended" (line 5).

LocationXYZ - contain center of mass of aggregated unit. Feeds "Location of Attack Grid" (line 6)

} end repeat

Perceived Status PDU

Number and types of equipment involved in attack are reported here.

CommoSite, Host, and ID - set up in Entity Commo PDU

CommoPartID - incremented from last CommoPartID

EntitySite, Host, ID, and TrackNum - contains identification for attacking unit with respect to Monitor Unit

Fidelity - probably contains 3 for non-cooperating

EffectiveTime - contains time of intel

NumBelongings - contains number of distinct types of attack means to be reported

repeat {

EntityTypeRecord - defines type of equipment firing

PresentAmount - number of this type involved in attack

TotalLosses - typically contains 0 for N/A

TotalGains - typically contains 0 for N/A

LossRate - typically contains 0 for N/A

GainRate - typically contains 0 for N/A

} for each resource

For each repetition of Resource status, one phrase of "Number and Nature" (line 7) is filled in.

Perceived Tactics PDU

CommoSite, Host, and ID - set up in Entity Commo PDU

CommoPartID - incremented from last CommoPartID

EntitySite, Host, ID, and TrackNum - contains identification of enemy unit with respect to Monitor Unit

Fidelity - probably contains 3 for non-cooperative

NumTactics - contains number of perceived attacker tactics, probably 1

repeat {

EffectiveTime - contains time of tactics perception

CorrelationSite, Host, and Code - identifies a tactic which the enemy unit is believed to be operating under. (i.e. Barrage, Registration, etc). Feeds "Nature of Fire" (line 8). The tactics would be displayed as: "Firing tactics for unit at <DTG>: <tactic string>, <tactic string>, ..., <tactic string>.

} for each tactic

Perceived Status PDU

All battle resources affected by attack will be reported with this PDU.

CommoSite, Host, and ID - set up in Entity Commo PDU

CommoPartID - incremented from last CommoPartID

EntitySite, Host, ID, and TrackNum - contains identification for unit submitting report with respect to Monitor Unit

Fidelity - contains 1 for self

EffectiveTime - contains time of intel

NumBelongings - contains number of distinct types of resources to be reported

repeat {

EntityTypeRecord - defines type of resources: personnel, equipment, etc.

PresentAmount - amount of this type of resource on hand

TotalLosses - amount of losses sustained to this type of resource

TotalGains - amount of gains for this type of resource

LossRate - num losses per second

GainRate - num gains per second

} for each resource

For each repetition of Resource status, one phrase for "Damage" (line 11) is filled in. Can be reported as percentages or present amounts

<u>**Instructions for Contact Report creation:**</u>

The PDUs expected in a Contact Report are listed below. For each, the fields of the PDU are defined with reference to Contact Reports.

"Direction" (line3) can be found by finding the azimuth from the observed unit location and the reporting unit location (from last perception update).

Entity Commo PDU

Commo Site, Host, and ID - unique to this report

CommoPartID - incremented from last CommoPartID

RelatedCommo Site, Host, and ID - probably 0's since Contact Reports aren't usually correlated with previous reports.

fromEntitySite, Host, ID, and TrackNum - contains id for "Observer" (line 1) with respect to Monitor Unit

CorrelationSite, Host, and Code - contains code for Contact Report

TimeOfReport - contains time of contact

NumPDUs - contains number of PDUs to be sent for this report

NumRecipients - probably only sent to 1 (superior)

repeat {

toEntitySite, Host, ID, and TrackNum - corresponds to superior unit.

} for each recipient

Entity Locations PDU

A Entity Locations PDU will be used to define the type of unit contacted and its location.

CommoSite, Host, and ID - set up in Entity Commo PDU

CommoPartID - incremented from last CommoPartID

NumEntities - contains 1

repeat {

EntitySite, Host, and ID - contains identification for the Monitor Unit.

TrackNum - the number local to the MonitorUnit which is used to represent the entity contacted. This value may be used to distinguish among aggregations of the same entity type (if Flag C, below, is 0). This number is not needed for the Contact Report

PercepFlags - A contains 0 for N/A

B probably contains 3 for non-cooperative data

C probably contains 2 for OrgName unknown

D..I contain 0 for N/A

EntityAppearance - probably all 0s, possibly dead is set

EntityOrgCode Site, Host, and Code - contains semantic code corresponding to org name if Flag C (above) is 1.

EntityType - corresponding type id from DIS. This type feeds "Type of unit" (line 2).

Timestamp - contains time of last perception update.

LocationXYZ - contain center of mass of observer unit.

} for NumEntities

Instructions for REQUEST FOR ENGINEERS creation:

The PDUs expected in a REQUEST FOR ENGINEERS are listed below. For each, the fields of the PDU are defined with reference to REQUEST FOR ENGINEERS.

Entity Commo PDU

Commo Site, Host, and ID - unique to this report

CommoPartID - incremented from last CommoPartID

RelatedCommo Site, Host, and ID - probably 0's since Requests For Engineers aren't usually correlated with previous reports.

fromEntitySite, Host, ID, and TrackNum - contains id for "Unit submitting report" (line 1) with respect to Monitor Unit

CorrelationSite, Host, and Code - contains code for Request For Engineers

TimeOfReport - contains "DTG of Report" (line2)

NumPDUs - contains number of PDUs to be sent for this report

NumRecipients - probably only sent to 1 (superior)

repeat {

toEntitySite, Host, ID, and TrackNum - corresponds to superior unit.

} *for each recipient*

General Purpose Request PDU

CommoSite, Host, and ID - set up in Entity Commo PDU

WhenFlag - 1, 2, or 3 depending on situation

Priority - 1 for Flash
2 for Immediate
3 for Priority
4 for Routine

RequestTime - contains time of request

DeliveryTime - time to meet at Delivery Point

DeliveryXYZ - location to meet for coordination

Correlation Site, Host, and Code - probably no tactic associated here.

NumRequests - number of things requested probably only 1

repeat {

EntityTypeRecord - identifies type of thing needed - here Engineers

Shell, Fuze - 0s for N/A

Quantity - specifies amount needed

} for NumRequests

<u>**Instructions for Call For Fire creation:**</u>

The PDUs expected in a Call For Fire are listed below. For each, the fields of the PDU are defined with reference to Call For Fire.

Entity Commo PDU

Commo Site, Host, and ID - unique to this report

CommoPartID - incremented from last CommoPartID

RelatedCommo Site, Host, and ID - probably 0's since Call For Fire isn't usually correlated with previous reports.

fromEntitySite, Host, ID, and TrackNum - contains id for entity requesting fire support. Feeds "Requesting Unit" (line 1) with respect to Monitor Unit

CorrelationSite, Host, and Code - contains code for Call For Fire

TimeOfReport - contains "DTG of Report" (line2)

NumPDUs - contains number of PDUs to be sent for this report

NumRecipients - probably only sent to 1 (superior or attached fire capable unit or FDC)

repeat {

toEntitySite, Host, ID, and TrackNum - corresponds to unit to receive fire request.

} for each recipient

General Purpose Request PDU

CommoSite, Host, and ID - set up in Entity Commo PDU

CommoPartID - incremented from last CommoPartID

WhenFlag - 1, 2, or 3 depending on situation. Feeds "Method of Fire Control" (line 7).

Priority - 1 for Flash

2 for Immediate

3 for Priority

4 for Routine

Feeds "Priority" (line 2)

RequestTime - contains time of request

DeliveryTime - time to fire if WhenFlag = 3. If valid, feeds "Method of Fire Control" (line 7)

DeliveryXYZ - location to fire at. Feeds "Target Location" (line 4). This value is always a grid coordinate (WGS84).

CorrelationSite, Host, and Code - identifies a tactic for AdjustFire, FireForEffect, or other fire tactic. Feeds "Type of Fire" (line 3).

NumRequests - 1 for one type of firing

repeat {

EntityTypeRecord - identifies type of guns

Shell, Fuze - shell and fuze to use or 0 for FDC to figure it.

Quantity - specifies number of bursts for this ammo.

} for NumRequests

Entity Locations PDU

A Entity Locations PDU will be used to define the type of target unit and its location.

CommoSite, Host, and ID - set up in Entity Commo PDU

CommoPartID - incremented from last CommoPartID

NumEntities - probably one

repeat {

EntitySite, Host, and ID - contains identification for the Monitor Unit.

TrackNum - the number local to the MonitorUnit which is used to represent the entity targeted. This value may be used to distinguish among aggregations of the same entity type (if Flag C below is 0). This number is not needed for the Contact Report.

PercepFlags - A contains 0 for N/A

B probably contains 3 for non-cooperative data

C probably contains 2 for OrgName unknown

D..I contain 0 for N/A

EntityAppearance - probably all 0s, possibly dead is set,

EntityOrgCode Site, Host, and Code - contains semantic code corresponding to org name if Flag C (above) is 1.

EntityType - corresponding type id from DIS. This entity type is used for "Target Description" (line 5).

Timestamp - contains time of last perception update.

LocationXYZ - contain center of mass of aggregated unit. Probably same as location fire is called for.

} for NumEntities

3. DETAILED REQUIREMENTS

The material for in this section should be suitable for insertion in Section 5 "Detailed Requirements" of the DIS protocol standard. If this is a SIMNET protocol extension, the information should still be in this format.

3.1. INTRODUCTION

This section defines the PDUs and their fields.

3.2. REPRESENTATION OF DATA

This paragraph will note any variation with the base standard's representation of data.

3.2.1. Enumerated Radix 10

TBS

3.3. BASIC DATA TYPES AND RECORDS

This paragraph will define any basic data types or records that are added/changed by this protocol extension. Data types and records should be defined in this section if they are referenced multiply or they may be useful outside of this protocol extension.

3.3.1. Correlation Identifier Record

3.3.1.1. Simulation Application Record.

Note: This definition should replace paragraph 5.3.8.1 in [IST 1993]

A simulation application's address shall be specified by a Simulation Application Record. A Simulation Application Record shall consist of the site identification number and the host identification number. These fields are described in 3.3.1.1. and 3.3.1.2. The Simulation Application Record is represented in Fig 3-1.

3.3.1.1. Site Identifier.

Each DIS site shall be assigned a unique site identifier. No site shall be assigned an ID containing all zeros or all ones. The mechanism by which site IDs are assigned is outside the scope of this standard. This identifier shall be specified by a 16-bit unsigned integer.

3.3.1.2. Application Identifier.

Each application at a DIS site shall be assigned a host identifier unique within that site. No application shall be assigned an ID containing all zeros or all ones. The mechanism by which application IDs are assigned is outside the scope of this standard. This identifier shall be specified by a 16-bit unsigned integer.

Site Identifier	16-bit unsigned integer
Applicaion Identifier	16-bit unsigned integer

Fig 3-3-1

Simulation Application Record

3.3.1.2. Correlation Identifier

Simulation Application Record	32 bits
Correlation Code	32-bit unsigned integer

Fig 3-3-2

Correlation Identifier Record

3.4. PROTOCOL DATA UNITS FOR PROTOCOL EXTENSIONS

3.4.1. Basic Data Types and Records

This paragraph will define any basic data types or records that are used by this protocol extension. Normally this paragraph contains only information data types and records that are used by multiple PDUs or multiple times with a PDU.

3.4.2. List of PDUs in Protocol Extension

The C3I protocol extension for

- PDUs FOR INITIALIZATION:
  - Correlation
  - Perception Control
  - Non-Point Control Measures
  - Point Control Measures
  - Point Control Measure with Relations
- PDUs FOR COMMUNICATIONS:
  - Entity Communication
  - Ctrl Measures for Commo
  - Entity Locations
  - Task Organization
  - General Purpose Request
  - Perceived Status
  - Perceived Tactics
  - Incident / Situation

PDUs FOR SIMULATION CONTROL:

(Re)Start

Admin Request

Impending Admin Action

3.4.3. Incident PDU

Incidents and situations shall be communicated by using the Incident PDU. The Incident PDU shall contian the following fields:

1) PDU Header - The PDU Header shall be represented by the PDU Header Record (see 5.3.15. of [IST 1993].
2) From entity ID - unique initiating entity identifier
3) To entity ID - unique receiving entity identifier
4) About entity ID - unique indirect object entity identifier
5) Scenerio Time - effective scenario time for the incident/situation
6) StringLength - length of correlation string
7) Incidentcode - code which describes an incident happening between the given entities. This code is normally defined by correlation PDUs.

The Incident PDU is represented in Figure 3-4-1.

Field Size (bits)	Incident/Situation PDU FIELDS	
32	PROTOCOL HEADER	Protocol Version - 8 bit unsigned integer Exercise ID - 8 bit unsigned integer PDU Type - 8 bit enumeration Length - 8 bitunsigned integer
48	FROM ENTITY ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Entity - 16 bit unsigned integer
48	TO ENTITY ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Entity - 16 bit unsigned integer
48	ABOUT ENTITY ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Entity - 16 bit unsigned integer
16	PADDING	16 bit unused
64	SCENERIO TIME	64 bit unsigned integer
32	INCIDENT CODE	32 bit unsigned integer

Figure 3-4-1. Incident/Situation PDU

3.4.4. Correlation PDU

This format is used to relate a code to a character string and to define any supporting information. The Correlation PDU shall contian the following fields:

1) PDU Header - The PDU Header shall be represented by the PDU Header Record (see 5.3.15. of [IST 1993].
2) Correlation ID - unique identifier of the correlation that is being defined. See 3.3.1.2.
3) SupportingInfoOrder - 0 for correlation code, >0 order for supporting information
4) Scenerio Time - effective scenario time for correlation
5) StringLength - length of correlation string
6) CorrelationString - character string describing thing being correlated or the supporting information

The Correlation PDU is represented in Figure 3-4-2.

	Field Size (bits)	Correlation PDU FIELDS	
	32	PROTOCOL HEADER	Protocol Version - 8 bit unsigned integer Exercise ID - 8 bit unsigned integer PDU Type - 8 bit enumeration Length - 8 bitunsigned integer
	64	CORRELATION ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Correlation Code - 32 bit unsigned integer
	32	SUPPORTING INFO ORDER	32 bit unsigned integer
	64	SCENERIO TIME	64 bit unsigned integer
	32	STRING LENGTH	32 bit unsigned integer
repeat STRING LENGTH times	n x 8	CORRELATION STRING	8 bit unsigned integer

Figure 3-4-2. Correlation PDU

3.4.5. Perception Control PDU

This PDU will be used to control perception reports for entities. The Perception Control PDU shall contian the following fields:

1) PDU Header - This field shall be represented by the PDU Header Record (see 5.3.15. of [IST 1993].
2) Entity Identification - This field shall be represented by the Entity Identification Record (see 5.3.8. of [IST 1993].
3) Perception Code - Flag defining perception. See 4.2.3.

5) Timestamp - Scenario time for perception control

The Perception Control PDU is represented in Figure 3-4-3.

Field Size (bits)	PERCEPTION CONTROL PDU FIELDS	
32	PROTOCOL HEADER	Protocol Version - 8 bit unsigned integer Exercise ID - 8 bit unsigned integer PDU Type - 8 bit enumeration Length - 8 bitunsigned integer
48	ENTITY ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Entity - 16 bit unsigned integer
16	PADDING	16 bit unused
32	PERCEPTION CODE	32 bit unsigned integer
64	TIMESTAMP	64 bit unsigned integer

Figure 3-4-3. Perception Control PDU

3.4.6. Entity Communication PDU

This PDU will be used to describe communication between entities. The Entity Communication PDU shall contian the following fields:

1) PDU Header - The PDU Header shall be represented by the PDU Header Record (see 5.3.15. of [IST 1993].
2) Communication ID - Unique identifier of the communication event and shall be represented by the Thing Identification Record defined in 3.3.3.
3) CommoPartID - Always 0 for first part of commo
4) RelatedCommoID - Unique ID for related commo
5) From Entity Identification - This field identifies the entity initiating a communication and shall be represented by the Entity Identification Record (see 5.3.8. of [IST 1993].
6) From Entity Track Num - Monitor Unit's Track number for "from" unit
7) CorrelationID - correlation code identifying type of communication. See 3.3.1.2.
8) TimeOfReport - Time report was created
9) NumPDUs - Number of PDUs for this commo
10) Number Recipients- Number of recipient entities
11) To Entity ID - Unique identifier for each commo destination entity
12) To Entity Track Num - Monitor Unit's Track number for "to" unit

The Entity Communication PDU is represented in Figure 3-4-4.

	Field Size (bits)	ENTITY COMMUNICATION PDU FORMAT	
	32	PROTOCOL HEADER	Protocol Version - 8 bit unsigned integer Exercise ID - 8 bit unsigned integer PDU Type - 8 bit enumeration Length - 8 bitunsigned integer
	48	COMMO ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Commo - 16 bit unsigned integer
	16	COMMO PART ID	16 bit unsigned integer
	48	RELATED COMMO ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Commo - 16 bit unsigned integer
	48	FROM ENTITY ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Entity - 16 bit unsigned integer
	16	FROM ENTITY TRACK	16 bit unsigned integer
	16	PADDING	16 bit unused
	64	CORRELATION ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Correlation Code - 32 bit unsigned integer
	64	TIME OF REPORT	64 bit unsigned integer
	32	NUMBER PDU	32 bit unsigned integer
	32	NUMBER RECIPIENT	32 bit unsigned integer
repeat NUMBER RECIPIENT times	48	TO ENTITY ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Entity - 16 bit unsigned integer
	16	FROM ENTITY TRACK	16 bit unsigned integer

Figure 3-4-4 Entity Communication PDU

3.4.7. Task Organization

This PDU will be used to describe the subordinates of a single entity.. The Task Organization PDU shall contian the following fields:

1) PDU Header - The PDU Header shall be represented by the PDU Header Record (see 5.3.15. of [IST 1993].

2) Communication ID Unique identifier of the communication event and shall be represented by the Thing Identification Record defined in 3.3.3.

3) Communication Part ID Identifies this PDU uniquely within this communication event specified by the Communication ID.

4) Entity Identification - This field identifies the entity initiating a communication and shall be represented by the Entity Identification Record (see 5.3.8. of [IST 1993].

5) Entity Track Number - Monitor Unit's Track Number for entity.

6) CorrelationID - Code identifying type of command chain. See 3.3.1.2.

7) Number Subordinates - number of subordinates following

8) Scenerio Time - effective scenario time for task organization

9) Subodinate Entity ID - This field identifies the entity initiating a communication and shall be represented by the Entity Identification Record (see 5.3.8. of [IST 1993].

10) Subodinate Track Number - Monitor Unit's Track Number for entity.

The Task Organization PDU is represented in Figure 3-4-5.

	Field Size (bits)	TASK ORGANIZATION PDU FIELDS	
	32	PROTOCOL HEADER	Protocol Version - 8 bit unsigned integer Exercise ID - 8 bit unsigned integer PDU Type - 8 bit enumeration Length - 8 bitunsigned integer
	48	COMMO ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Thing - 16 bit unsigned integer
	16	COMMO PART ID	16 bit unsigned integer
	48	MONITOR ENTITY ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Entity - 16 bit unsigned integer
	16	MONITOR ENTITY TRACK	16 bit unsigned integer
	64	CORRELATION ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Correlation Code - 32 bit unsigned integer
	32	NUMBER SUBORDINATES	32 bit unsigned integer
	64	SCENERIO TIME	64 bit unsigned integer
repeat NUMBER SUBORDINATE times	48	SUBORDINATE ENTITY ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Entity - 16 bit unsigned integer
	16	SUBORDINATE ENTITY TRACK	16 bit unsigned integer

Figure 3-4-5. Task Organization PDU

3.4.8. (Re)Start PDU

This PDU will be issued just prior to starting the simulation or restarting the simulation (after being paused). The (Re)Start PDU shall contian the following fields:

1) PDU Header - The PDU Header shall be represented by the PDU Header Record (see 5.3.15. of [IST 1993].
2) ScenarioTimeUnits Units of Measure for scenario time
3) ScenarioStartTime Beginning time for scenario
4) TimeCoordinate x WGS84 X for scenario start time
5) TimeCoordinate y WGS84 Y for scenario start time
6) TimeCoordinate z WGS84 Z for scenario start time
7) GameStartTime Game Time at (re)start
8) CountToStart Delta wall clock time until (re)start (in seconds)
9) RealTimeMult Execution speed as fraction of wall clock time

The (Re)Start PDU is represented in Figure 3-4-6.

Field Size (bits)	(RE)START PDU FORMAT	
32	PROTOCOL HEADER	Protocol Version - 8 bit unsigned integer Exercise ID - 8 bit unsigned integer PDU Type - 8 bit enumeration Length - 8 bitunsigned integer
32	SCENARIO TIME UNITS	32 bit unsigned integer
64	SCENARIO START TIME	64 bit unsigned integer
64	TIME COORDINATE X	64 bit unsigned integer
64	TIME COORDINATE Y	64 bit unsigned integer
64	TIME COORDINATE Z	64 bit unsigned integer
64	GAME START TIME	64 bit unsigned integer
64	COUNT TO START	64 bit unsigned integer
64	REAL TIME MULT	64 bit unsigned integer

Figure 3-4-6. (Re)Start PDU

3.4.9. Admin Request PDU

This PDU will be sent to the Master Controller (CGF Engine) whenever changes to the execution parameters of the simulation are desired. The Admin Request PDU shall contian the following fields:

1) PDU Header - The PDU Header shall be represented by the PDU Header Record (see 5.3.15. of [IST 1993].
2) ActionRequested - Administrative Request. See 4.2.4.
3) EffectiveGameTime - Effective absolute game time for request
4) CountToStart - Requested delta wall clock time until change
5) RealTimeMult - New real time multiple or 0

The Admin Request PDU is represented in Figure 3-4-7.

Field Size (bits)	ADMIN REQUEST PDU FORMAT	
32	PROTOCOL HEADER	Protocol Version - 8 bit unsigned integer Exercise ID - 8 bit unsigned integer PDU Type · 8 bit enumeration Length - 8 bitunsigned integer
32	ACTION REQUESTED	32 bit unsigned integer
64	EFFECTIVE GAME TIME	64 bit unsigned integer
64	COUNT TO START	64 bit unsigned integer
64	REAL TIME MULT	64 bit unsigned integer

Figure 3-4-7. Admin Request PDU

3.4.10. Impending Admin Action PDU

This PDU will be issued by the Master Controller (CGF Engine) whenever changes to the execution parameters of the simulation are going into effect. The Impending Admin Action PDU shall contian the following fields:

1) PDU Header - The PDU Header shall be represented by the PDU Header Record (see 5.3.15. of [IST 1993].
2) ImpendingAction - Impending Administrative Action. See 4.2.5.
3) EffectiveGameTime - Effective absolute game time for request
5) RealTimeMult - New real time multiple or 0

The Impending Admin Action PDU is represented in Figure3-4-8.

Field Size (bits)	IMPENDING ADMIN REQUEST PDU FORMAT	
32	PROTOCOL HEADER	Protocol Version - 8 bit unsigned integer Exercise ID - 8 bit unsigned integer PDU Type - 8 bit enumeration Length - 8 bitunsigned integer
32	IMPENDING ACTION	32 bit unsigned integer
64	EFFECTIVE GAME TIME	64 bit unsigned integer
64	REAL TIME MULT	64 bit unsigned integer

Figure 3-4-8. Impending Admin Action PDU

3.4.11. General Purpose Request PDU

This PDU shall communicate general purpose requests. The General Purpose Request PDU shall contian the following fields:

1) PDU Header - The PDU Header shall be represented by the PDU Header Record (see 5.3.15. of [IST 1993].
2) CommoID - Unique identifier for commo
3) CommoPartID - Identifies this PDU uniquely within this commo
4) Priority - Priority of Request (0..65535)
5) WhenFlag - When to perform request. See 4.2.2.
6) Correlation ID - Site identifier for commo origination
7) RequestTime - GameTime of request
8) DeliveryTime - GameTime to perform request
9) Delivery Location - Location to deliver to or meet at
10) NumRequests - Number of requests enumerated below
11) EntityType - Define the type of resource needed
12) Fuze - Fuze type if needed
13) Warhead - Warhead type if needed
14) Quantity - Number of this type needed

The General Purpose Request PDU is represented in Figure 3-4-9.

	Field Size (bits)	GENERAL PURPOSE REQUEST PDU FIELDS	
	32	PROTOCOL HEADER	Protocol Version - 8 bit unsigned integer Exercise ID - 8 bit unsigned integer PDU Type - 8 bit enumerauon Length - 8 bitunsigned integer
	48	COMMO ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Thing - 16 bit unsigned integer
	16	COMMO PART ID	16 bit unsigned integer
	16	PRIORITY	16 bit unsigned integer
	16	WHEN FLAG	16 bit unsigned integer
	64	CORRELATION ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Correlation Code - 32 bit unsigned integer
	64	REQUEST TIME	64 bit unsigned integer
	64	DELIVERY TIME	64 bit unsigned integer
	192	DELIVERY LOCATION	X-Component - 64 bit floating point Y-Component - 64 bit floating point Z-Component - 64 bit floating point
	32	NUMBER REQUESTS	32 bit unsigned integer
repeat NUMBER REQUESTS times	64	ENTITY TYPE	64 bit unsigned integer
	8	FUZE	8 bit enumerated
	8	WARHEAD	8 bit enumerated
	16	QUANTITY	16 bit unsigned integer

Figure 3-4-9. *General Purpose Request PDU*

3.4.12. Perceived Status PDU

The Perceived Status PDU shall contian the following fields:

1) PDU Header - The PDU Header shall be represented by the PDU Header Record (see 5.3.15. of [IST 1993].
2) CommoID - Unique identifier for commo on this site and host
3) CommoPartID - Identifies this PDU uniquely within this commo
4) EntityID - Unique ID for Monitor Unit on this site and host
5) EntityTrackNum - Monitor Unit's Track Number for entity
6) Fidelity - Fidelity of Data. See 4.2.1.
7) EffectiveTime - Effective Game Time for status report
8) NumBelongings - Number battle resources types reported

9) EntityType - EntityType Record describing battle resource
10) PresentAmt - quantity of battle resource on hand
11) TotalLosses - absolute value of losses
12) TotalGains - absolute value of gains
13) LossRate - losses per second
14) GainRate - gains per second

The Perceived Status PDU is represented in Figure 3-4-10.

Field Size (bits)	PERCEIVED STATUS PDU FORMAT	
32	PROTOCOL HEADER	Protocol Version - 8 bit unsigned integer Exercise ID - 8 bit unsigned integer PDU Type - 8 bit enumeration Length - 8 bitunsigned integer
48	COMMO ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Thing - 16 bit unsigned integer
16	COMMO PART ID	16 bit unsigned integer
48	MONITOR ENTITY ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Entity - 16 bit unsigned integer
16	MONITOR ENTITY TRACK	16 bit unsigned integer
16	FIDELITY	16 bit enumeration
64	EFFECTIVE TIME	64 bit unsigned integer
32	NUMBER BELONGINGS	32 bit unsigned integer
64	ENTITY TYPE	64 bit enumerated
64	ON HAND	64 bit enumerated
32	TOTAL LOSSES	32 bit floating point
32	TOTAL GAINS	32 bit floating point
32	LOSS RATE	32 bit floating point
32	GAIN RATE	32 bit floating point

repeat NUMBER BELONINGS times

Figure 3-4-10. Perceived Status PDU

3.4.13. Entity Locations PDU

This PDU will be used to describe entity locations, etc. The Entity Locations PDU shall contian the following fields:

1) PDU Header - The PDU Header shall be represented by the PDU Header Record (see 5.3.15. of [IST 1993].
2) CommoID - Unique identifier for commo on this site and host
3) CommoPartID - With commo information forms unique id for this PDU

4) NumEntities - Number of Entity locations defined
5) EntityID - Unique ID for Monitor Unit on this site and host
6) TrackNum - Monitor Unit's Track Number for related entity
7) PercepFlags - Perception flags ABCDEFGHI (radix 10)
8) EntityAppear. - Entity Appearance Flag
9) Correlation ID - Correlation Code for Entity Organization
10) EntityType - (same as DIS) used to describe type of thing reported
11) Timestamp - effective game time for Perception
11) Origin Location - origin Z location

The Entity Locations PDU is represented in Figure 3-4-11.

	Field Size (bits)	ENTITY LOCATIONS PDU FORMAT	
	32	PROTOCOL HEADER	Protocol Version - 8 bit unsigned integer Exercise ID - 8 bit unsigned integer PDU Type - 8 bit enumeration Length - 8 bitunsigned integer
	48	COMMO ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Commo - 16 bit unsigned integer
	16	COMMO PART ID	16 bit unsigned integer
	32	NUMBER ENTITIES	32 bit unsigned integer
repeat NUMBER ENTITIES times	48	MONITOR ENTITY ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Entity - 16 bit unsigned integer
	16	MONITOR ENTITY TRACK	16 bit unsigned integer
	32	PERCEPTION FLAG	32 bit radix 10
	32	ENTITY APPEARANCE	32 bit enumeration
	64	CORRELATION ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Correlation Code - 32 bit unsigned integer
	16	PADDING	16 bit unsigned integer
	64	ENTITY TYPE	64 bit unsigned integer
	64	TIMESTAMP	64 bit unsigned integer
	192	ORIGIN LOCATION	X-Component - 64 bit floating point Y-Component - 64 bit floating point Z-Component - 64 bit floating point

Figure 3-4-11. Entity Locations PDU

3.4.14. Non-Point Control Measures PDU

This PDU will be used to describe control measures consisting of multiple points. The Non-Point Control Measures PDU shall contian the following fields:

1) PDU Header - The PDU Header shall be represented by the PDU Header Record (see 5.3.15. of [IST 1993].
2) CtrlMeasID - Unique identifier for control measure
3) Correlation ID - correlation code for tactic to be associated with ctrl
4) Priority - (unsigned) Priority of effort/position
5) ShapeFlag - Type of Shape
6) EntityType - (same as DIS) used to describe type of thing reported
7) Timestamp - effective game time for control measure
8) Origin Location - origin Z location
9) NumPoints - number of points following to define Ctrl Measure
10) Location - relative location coordinate of subsequent point

The Non-Point Control Measures PDU is represented in Figure3-4-12.

	Field Size (bits)	NON-POINT CTRL MEASURES PDU FORMAT	
	32	PROTOCOL HEADER	Protocol Version - 8 bit unsigned integer Exercise ID - 8 bit unsigned integer PDU Type - 8 bit enumeration Length - 8 bitunsigned integer
	48	CTRL MEASURE ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Entity - 16 bit unsigned integer
	16	PADDING	16 bit unused
	64	CORRELATION ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Correlation Code - 32 bit unsigned integer
	16	PRIORITY	16 bit unsigned integer
	64	SHAPE FLAG	64 bit unsigned integer
	64	ENTITY TYPE	64 bit unsigned integer
	64	TIMESTAMP	64 bit unsigned integer
	192	ORIGIN LOCATION	X-Component - 64 bit floating point Y-Component - 64 bit floating point Z-Component - 64 bit floating point
	32	NUMBER POINTS	32 bit unsigned integer
repeat NUMBER POINTS times	192	ORIGIN LOCATION	X-Component - 64 bit floating point Y-Component - 64 bit floating point Z-Component - 64 bit floating point

Figure 3-4-12. Non-Point Control Measures PDU

3.4.15. Point Control Measures PDU

This PDU will be used to describe single point control measures. The Point Control Measures PDU shall contian the following fields:

1) PDU Header - The PDU Header shall be represented by the PDU Header Record (see 5.3.15. of [IST 1993].
2) NumCtrlMeasures - Number of single point control measures in this PDU
3) CtrlMeasID - Unique identifier for control measure
4) Correlation ID - Correlation code for tactic associated with ctrl.
5) Priority - (unsigned) Priority of effort/position
6) ShapeFlag - Type of Shape
7) EntityType - (same as DIS) used to describe type of thing reported
8) Timestamp - effective game time for control measure
9) Location - origin location

The Point Control Measures PDU is represented in Figure 3-4-13.

<table>
<tr><th></th><th>Field Size (bits)</th><th colspan="2">POINT CTRL MEASURES PDU FORMAT</th></tr>
<tr><td></td><td>32</td><td>PROTOCOL HEADER</td><td>Protocol Version - 8 bit unsigned integer
Exercise ID - 8 bit unsigned integer
PDU Type - 8 bit enumeration
Length - 8 bitunsigned integer</td></tr>
<tr><td></td><td>32</td><td>NUMBER CTRL MEASURES</td><td>32 bit unsigned integer</td></tr>
<tr><td rowspan="7">repeat NUMBER CTRL MEASURES times</td><td>48</td><td>CTRL MEASURE ID</td><td>Site - 16 bit unsigned integer
Application - 16 bit unsigned integer
Entity - 16 bit unsigned integer</td></tr>
<tr><td>16</td><td>PADDING</td><td>16 bit unused</td></tr>
<tr><td>64</td><td>CORRELATION ID</td><td>Site - 16 bit unsigned integer
Application - 16 bit unsigned integer
Correlation Code - 32 bit unsigned integer</td></tr>
<tr><td>16</td><td>PRIORITY</td><td>16 bit unsigned integer</td></tr>
<tr><td>64</td><td>SHAPE FLAG</td><td>64 bit unsigned integer</td></tr>
<tr><td>64</td><td>ENTITY TYPE</td><td>64 bit unsigned integer</td></tr>
<tr><td>64</td><td>TIMESTAMP</td><td>64 bit unsigned integer</td></tr>
</table>

Figure 3-4-13 Point Control Measures PDU

3.4.16. Point Control Measures with Relations PDU

This PDU will be used to describe a point control measure and to relate it to other previously defined control measures. The Point Control Measures with Relations PDU shall contian the following fields:

1) PDU Header - The PDU Header shall be represented by the PDU Header Record (see 5.3.15. of [IST 1993].
2) NumCtrlMeasures - Number of single point control measures in this PDU
3) CtrlMeasID - Unique identifier for control measure
4) Correlation ID - Correlation code for tactic associated with ctrl.
5) Priority - (unsigned) Priority of effort/position
6) ShapeFlag - Type of Shape
7) EntityType - (same as DIS) used to describe type of thing reported
8) Timestamp - effective game time for control measure
9) Location - origin location
10) NumRelations (32) number of Related control measures
11) CtrlMeasID (16) Unique identifier for related control measure

The Point Control Measures with Relations PDU is represented in Figure 3-4-14.

Field Size (bits)	CTRL MEASURES WITH RELATIONS PDU FORMAT	
32	PROTOCOL HEADER	Protocol Version - 8 bit unsigned integer Exercise ID - 8 bit unsigned integer PDU Type - 8 bit enumeration Length - 8 bitunsigned integer
48	CTRL MEASURE ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Entity - 16 bit unsigned integer
16	PADDING	16 bit unused
64	CORRELATION ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Correlation Code - 32 bit unsigned integer
16	PRIORITY	16 bit unsigned integer
64	ENTITY TYPE	64 bit unsigned integer
64	TIMESTAMP	64 bit unsigned integer
192	ORIGIN LOCATION	X-Component - 64 bit floating point Y-Component - 64 bit floating point Z-Component - 64 bit floating point
32	NUMBER RELATIONS	32 bit unsigned integer
repeat NUMBER REALATIONS times: 48	ORIGIN LOCATION	Site - 16 bit floating point Host - 16 bit floating point Relation - 16 bit floating point

Figure 3-4-14. Point Control Measures with Relations PDU

3.4.17. Ctrl Measures for Commo PDU

This format is used define which previously defined control measures should be used with this communication. The Ctrl Measures for Commo PDU shall contian the following fields:

1) PDU Header - The PDU Header shall be represented by the PDU Header Record (see 5.3.15. of [IST 1993].
2) CommoID - Unique identifier for commo on this site and host
3) CommoPartID - With commo information forms unique id for this PDU
4) NumCtrlMeasures - Number of control measures to use with commo
5) CtrlMeasID - Unique ID for control measure

The Ctrl Measures for Commo PDU is represented in Figure 3-4-15.

	Field Size (bits)	CTRL MEASURES FOR COMMO PDU FORMAT	
	32	PROTOCOL HEADER	Protocol Version - 8 bit unsigned integer Exercise ID - 8 bit unsigned integer PDU Type - 8 bit enumeration Length - 8 bitunsigned integer
	48	COMMO ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Commo - 16 bit unsigned integer
	16	COMMO PART ID	16 bit unsigned integer
	32	NUMBER CTRL MEASURES	32 bit unsigned integer
repeat NUMBER CTRL MEASURES times	48	TO ENTITY ID	Site - 16 bit unsigned integer Application - 16 bit unsigned integer Entity - 16 bit unsigned integer

Figure 3-4-15. Ctrl Measures for Commo PDU

4. ENUMERATED AND BIT ENCODED VALUES FOR USE WITH (PROTOCOL EXTENSION)

The material for in this section should be suitable for insertion in the "Enumerated and Bit Encoded Values for Use with Protocols for Distributed Interactive Simulation Applications" document that accompanies the base standard (see section 1.1). If this is a SIMNET protocol extension, the information should still be in this format.

4.1. UPDATED FIELDS

4.1.1. PDU Kind

This field is 4.3.1.6 of [IST 1993]. The following values were appended to this field by the protocol extension.

Field Value	PDU Kind
140	Incident/Situation
142	Correlation
144	Perception Control
145	Entity Communication
149	Task Organization
150	(Re)Start
151	Admin Request
152	Impending Admin Action
153	General Purpose
153	General Purpose Request
154	Perceived Status
155	Perceived Tactics
156	Entity Locations
157	Non-Point Control Measures
158	Point Control Measures
159	Point Control Measure with Relations
160	Ctrl Measures for Commo

4.1.2. Entity types.

This field is Section 6 of [IST 1993]. The following values were appended to this field by the protocol extension.

Entity types for entity kind "Supply" are given in 6.3.11 of [IST 1993]. The following **bold** values were appended to this field by the protocol extension.

Kind	Dom	Count	Cat	Scat	Spec
6 Supply					
	0 Unused				
		0 Unused			
			6 Personnel		
				0 Other	
				1 Dismounted Infantry	
				2 Engineers	
				3 TBD	
			7 Water		

This protocol extension introduced a new entity kind "Control Measure". The Entity Types for this entity kind are given in the following table.

Kind	Domain	Country	Service	Type	Measure	Enumeration
8 Control Measures						
	0 Unused					
		168 USA				
			0 Other			
			1 Army			
				1 Point		
					1 Contact Point	
					2 Coord./Control Point	
					3 Passage Point	
					4 Release Point	
					5 Start Point	
					6 Reference Point	
				2 Line		
					1 Phase Line	
					2 Boundary Line	
						1 Lateral
						2 Rear
					3 Fire Support Coordination Line	
					4 Line of Attack	
					5 Line of Departure	
					6 Route Line	
					7 Security Ops	
						1 Screen
						2 Guard
						3 Cover
					8 Axis of Advance	
					9 Direction of Attack	
					10 Line Departure is Line Attack	
				2 Area		
					1 Assembly Area	
					2 Objective	
					3 Pre-planned Fire Area	
					4 No Fire Area	
					5 Restrictive Fire Area	
					6 Obstacles	
						1 Minefield
						2 TBD

4.2. NEW FIELDS

4.2.1. Fidelity

The values presently defined for this field are as follows:

1 - Self

2 - Cooperative

3 - Non-Cooperative

4.2.2. When Flag

The values presently defined for this field are as follows:

1 - On Order

2 - When Ready

3 - At Delivery Time

4.2.3. Perception Code

The values presently defined for this field are as follows: ABCDEFGHI (radix 10)

A - Perception Control Code
- 0 - Stop sending perceptions
- 1 - Begin sending perceptions

B - Perception Report Type
- 0 - Snapshot of perceptions at time of request
- 1 - Continous, update perceptions until Perception Control of Stop is received.

C - Perception Type
- 1 - Monitor
- 2 - Control

D - Info Categories (bit specified is set when true)
- 0 - Perceptions
- 1 - Communications

E..I - Not Used

4.2.4. ActionRequested

The values presently defined for this field are as follows:

1 - Pause

2 - Change Real time multiple

3 - (Re)Start

4 - (Re)Start with new Real time multiple

4.2.5. ImpendingAction

The values presently defined for this field are as follows:

1 - Pause

2 - Change Real time multiple

4.2.6. ShapeFlag

1 - Point

2 - Line

3 - Horizontal Area

Appendix B2: Digital Message Communications Protocol Extension

1. DIGITAL MESSAGE COMMUNICATIONS PROTOCOL (DMCP)

Under AIRNET AeroModel & Weapons Model Conversion, the MCC Comanche Support and Digital Message/Communications Upgrade provides for the use of existing MCC functions within the AIRNET simulation for Comanche simulators, and to provide for digital messaging capability between the Fire Support Element, the Tactical Operations Center and the RAH-66 Comanche simulator(s).

1.1. BASE STANDARD

This was implemented as an extension of both the SIMNET 6.6.1 protocol [BBN, 1991] and the DIS Application Protocol [IEEE, 1993].

The Digital Message/Communications Segment is designed such that the executable code will be completely compatible with any and all digital message formats, with such messages and formats being described in separate data files. These data files will provide the executable with all the information it needs about the particular message type, including the name and number of the message, the message structure and the message length.

The MCC Digital Message/Communications Segment is intended to provide the capability to send and receive digital pre-formatted and free text messages between the MCC operator stations and Fire Support Element Console (FSEC) and the Comanche simulator(s).

1.2. APPLICABLE DOCUMENTS.

BBN, 1991 — Arthur Pope, Richard L. Schaffer. The SIMNET Network and Protocols BBN Report Number 7627, June 1991.

IEEE, 1993 — 1278-1993, Standard for Information Technology, Protocols for Distributed Interactive Simulation Applications, IEEE, New York, NY, March 1993

1.3. IMPLEMENTATION HISTORY.

Currently the Comanche Support and Digital Message/Communications Upgrade to the AIRNET MCC is <u>intended for use at only one site, Fort Rucker, Alabama</u>. No provision is made for portability to any other SIMNET facility. However, the upgrade was designed with portability and reusability in mind.

2. GENERAL REQUIREMENTS

TBS

3. DETAILED REQUIREMENTS

3.1. INTRODUCTION

The preliminary Protocol Data Unit for the Digital Message/Communications portion of the AIRNET upgrade consists of a specification, and a body. The specification contains data fields of the same kinds for each protocol data unit type, while the body contains that data which is specific to the particular type of PDU being transmitted.

3.2. REPRESENTATION OF DATA

TBS

3.3. BASIC DATA TYPES AND RECORDS

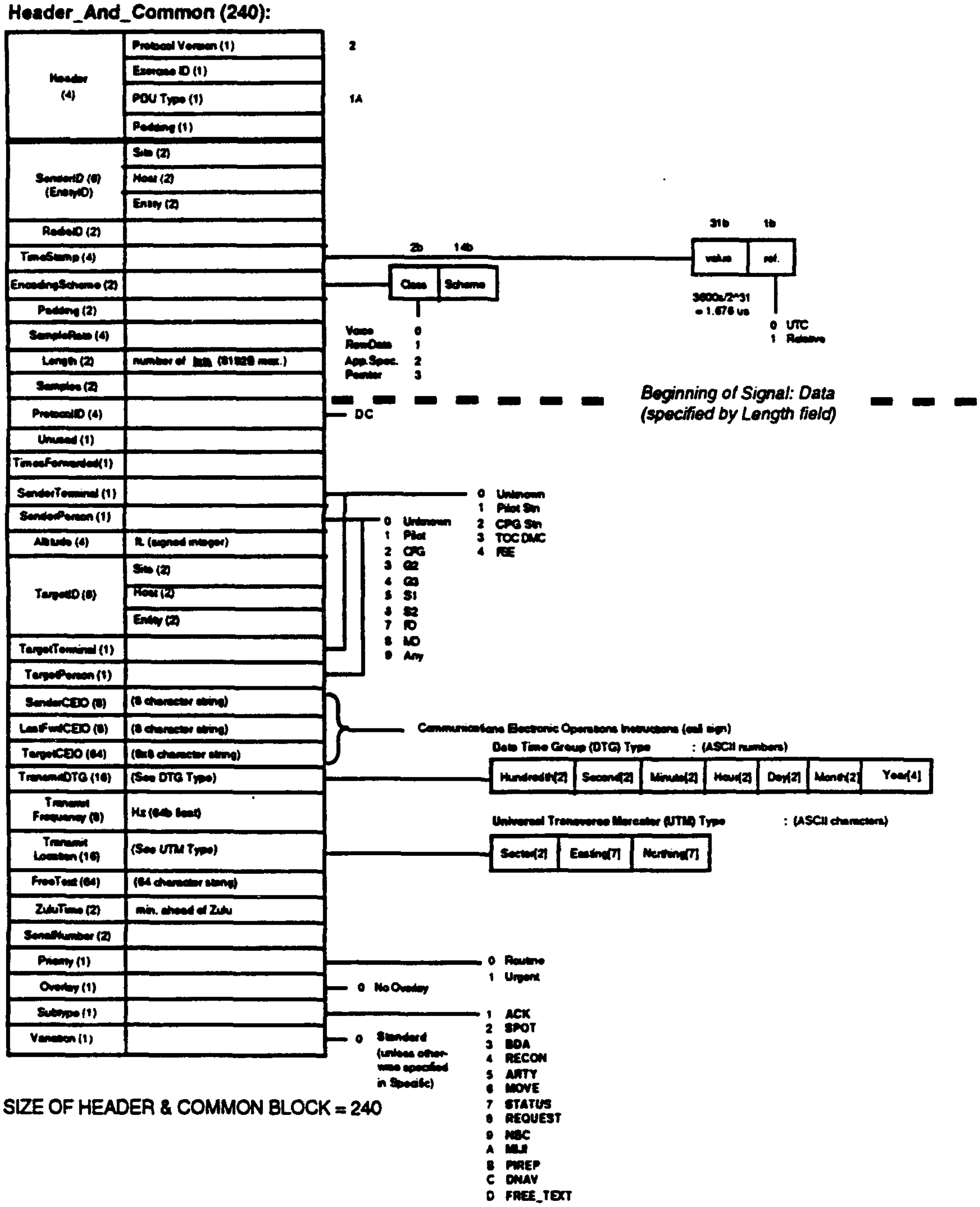
3.3.1. DIS Header

1A DMC (SIGNAL) PDU: DIS

Rev. Date: 1/6/93

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
fields are unsigned if not specified

Header_And_Common (240):

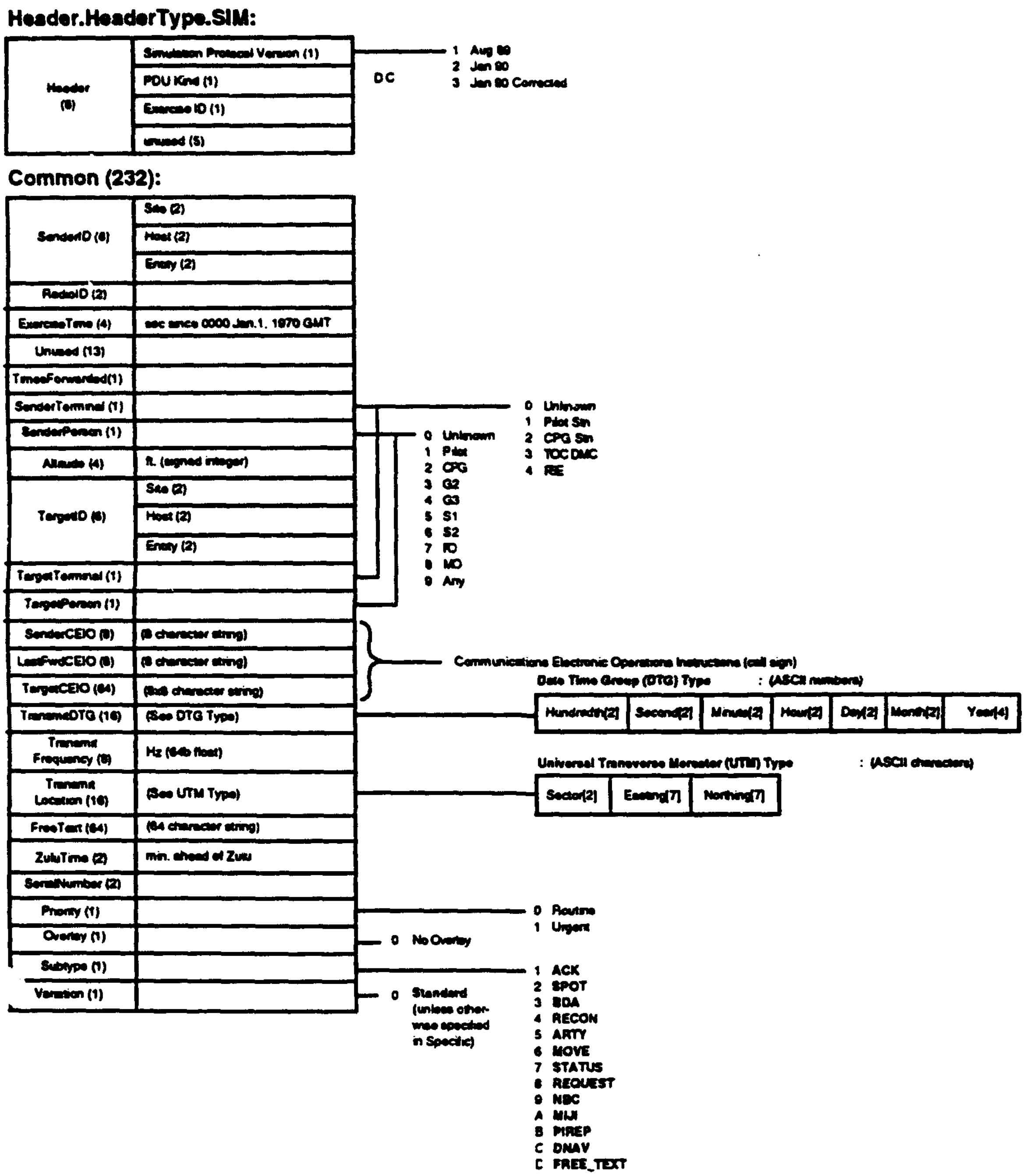


3.3.2. SIMNET Header

DC DMC PDU: Header & Common Block for Simnet

Rev. Date: 1/6/93

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
...ds are unsigned if not specified



SIZE OF HEADER & COMMON BLOCK = 240

3.4. LIST OF PDUS IN PROTOCOL EXTENSION

TBS

3.5. PROTOCOL DATA UNITS FOR PROTOCOL EXTENSIONS

3.5.1. DMC: ACK

DC DMC PDU Specific:
01 Subtype = ACK
00 Variation = Standard

Rev. Date:
11/23/92

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
fields are unsigned if not specified

Specific (104):

Original SenderID (6)	Site (2)
	Host (2)
	Entity (2)
OriginalSender Terminal (1)	
OriginalSender Person (1)	
AcknowledgeID (6)	Site (2)
	Host (2)
	Entity (2)
Acknowledge Terminal (1)	
Acknowledge Person (1)	
Original SenderCEIO (8)	(8 character string)
Acknowledge CEIO (8)	(8 character string)
OriginalDTG(16)	(See DTG Type)

TOTAL SIZE = 240 + 104 = 288 Bytes (includes Header & Common Block)

3.5.2. DMC: Spot

DC DMC PDU Specific:
02 Subtype = SPOT
00 Variation = Standard

Rev. Date:
11/23/92

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
fields are unsigned if not specified

Specific (88):

TimeSighted (16)	(See DTG Type)
Observer Location (16)	(See UTM Type)
TargetQuantity(2)	
Observer Intentions (1)	
TargetType (1)	
TargetActivity (1)	
TargetDirection (1)	
TargetSpeed (2)	(See Speed Type)
Target Location (16)	(See UTM Type)
TargetUnit (16)	(16 character string)
Unused (16)	

0 Not Entered
1 Other
2 Cont Mon
3 Engn
4 Hld
5 Rtn Instd
6 Obsv Tgt

0 Not Entered
1 Unknown
2 ADA
3 SAM
4 Tank
5 Whld
6 Trkd
7 Acft
8 Trps
9 Und

0 Not Entered
1 None
2 Stationary
3 Moving
4 Dug In
5 Retreating
6 Advancing
7 Attacking
8 Damaged
9 Killed
10 Mobile Kill

0 Not Entered
1 N
2 NE
3 E
4 SE
5 S
6 SW
7 W
8 NW
9 Unknown

MPH	Value
1b	15b

Speed Type :

TOTAL SIZE = 240 + 88 = 328 Bytes (includes Header & Common Block)

3.5.3. DMC: BDA

DC DMC PDU Specific:
03 Subtype = BDA
00 Variation = Standard

Rev. Date:
12/16/92

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
fields are unsigned if not specified

Specific (40):

StrikeTime Start (16)	(See DTG Type)
StrikeTime End (16)	(See DTG Type)
TargetCategory (1)	
TargetType (1)	
Targets Destroyed (2)	
Percent Coverage (1)	
Unused (3)	

0 Not Entered
1 Unknown
2 ADA
3 SAM
4 Tank
5 Whld
6 Trkd
7 Acft
8 Trps
9 Und

0 Not Entered
1 0 %
2 25 %
3 50 %
4 75 %
5 100 %

TOTAL SIZE = 240 + 40 = 280 Bytes (includes Header & Common Block)

3.5.4. DMC: RECON

3.5.4.1. DMC: RECON GRND RT

DC DMC PDU Specific:
04 Subtype = RECON
00 Variation = GND ROUTE

Rev. Date:
11/23/92

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
fields are unsigned if not specified

Specific (80): Subtype.GndRoute

EnemyActivity (1)	
Classification Formula (1)	
Unused (6)	
ClassFormInfo (32)	(32 character string)
RouteID (32)	(32 character string)
Leftover (8)	

EnemyActivity:
0 Not Entered
1 None
2 Stationary
3 Moving
4 Dug In
5 Retreating
6 Advancing
7 Attacking
8 Damaged
9 Killed
A Mobile Kill

Classification Formula:
0 Not Entered
1 W
2 Type
3 MLC
4 CHC
5 CB
6 Block

3.5.4.2. DMC: RECON AIR RTI

DC DMC PDU Specific:
04 Subtype = RECON
01 Variation = AIR ROUTE

Rev. Date:
11/23/92

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
fields are unsigned if not specified

Specific (80): Subtype.AirRoute

EnemyActivity (1)	
Obstacles (1)	
Unused (6)	
ClassFormInfo (32)	(32 character string)
RouteID (32)	(32 character string)
Leftover (8)	

EnemyActivity:
0 Not Entered
1 None
2 Stationary
3 Moving
4 Dug In
5 Retreating
6 Advancing
7 Attacking
8 Damaged
9 Killed
A Mobile Kill

Obstacles:
0 Not Entered
1 None
2 Keypad
3 ADA
4 Towr Ant
5 Wires
6 Terr
7 Trees
8 Bldg
9 Other

TOTAL SIZE = 240 + 80 = 320 Bytes (includes Header & Common Block)

3.5.4.3. DMC: RECON BRIDGE

D C DMC PDU Specific:
04 Subtype = RECON
02 Variation = BRIDGE

Rev. Date: 11/23/92

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
fields are unsigned if not specified

Specific (80): Subtype.Bridge

Type (1)	
Damage (2)	
Spans (1)	
ConstrMaterial (1)	
Unused (4)	(4 character string)
Length (4)	(4 character string)
Width (4)	(4 character string)
Heigth (4)	(4 character string)
Under (4)	(4 character string)
ConstrDescr (16)	(16 character string)
ID (32)	(32 character string)
SpanLength (4)	(4 character string)
LoadClass (4)	(4 character string)

Type:
0 Not Entered
1 Unknown
2 Truss
3 Girder
4 Beam
5 Slab
6 CLOS
7 Arch Op
8 Suspen
9 Floating
A Swing

Damage:
0 Not Entered
1 None
2 Light
3 Moderate
4 Severe
5 Destroyed

ConstrMaterial:
0 Not Entered
1 Unknown
2 Keypad
3 Metal
4 Stone
A Concrete

TOTAL SIZE = 240 + 80 = 320 Bytes (includes Header & Common Block)

3.5.4.4. DMC: RECON LZ/PZ

D C DMC PDU Specific:
04 Subtype = RECON
03 Variation = LZ_PZ

Rev. Date: 11/23/92

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
fields are unsigned if not specified

Specific (80): Subtype.LZ_PZ

ActivityLikely (1)	
Obstacles (1)	
Unused (8)	
Obstacles Descr (16)	(16 character string)
ID (16)	(16 character string)
LZ_Size (16)	(16 character string)
Axis (16)	(16 character string)
Leftover (8)	

ActivityLikely:
0 Not Entered
1 None
2 Expected
3 Possible
4 Unlikely

Obstacles:
0 Not Entered
1 None
2 Keypad
3 ADA
4 Towr Ant
5 Wires
6 Terr
7 Trees
8 Bldg
9 Other

TOTAL SIZE = 240 + 80 = 320 Bytes (includes Header & Common Block)

3.5.4.5. DMC: RECON BP/OI

D C DMC PDU Specific:
04 Subtype = RECON
04 Variation = BP_OP

Rev. Date: 11/23/92

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
fields are unsigned if not specified

Specific (80): Subtype.BP_OP

EnemyActivity (1)	
Obstacles (1)	
Unused (6)	
Obstacles Descr (16)	(16 character string)
ID (16)	(16 character string)
BP_Size (16)	(16 character string)
Axis (16)	(16 character string)
Leftover (8)	

0 Not Entered
1 None
2 Stationary
3 Moving
4 Dug In
5 Retreating
6 Advancing
7 Attacking
8 Damaged
9 Killed
A Mobile Kill

0 Not Entered
1 None
2 Keypad
3 ADA
4 Tank Ant
5 Wires
6 Terr
7 Trees
8 Bldg
9 Other

TOTAL SIZE = 240 + 80 = 320 Bytes (includes Header & Common Block)

3.5.4.6. DMC: RECON CROSSING

D C DMC PDU Specific:
04 Subtype = RECON
05 Variation = CROSSING

Rev. Date: 11/23/92

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
fields are unsigned if not specified

Specific (80): Subtype.Crossing

BankSlopeEntry (4)	(4 character string)
BankSlopeExit (4)	(4 character string)
CrossingLength (4)	(4 character string)
CrossingWidth (4)	(4 character string)
CrossingDepth (4)	(4 character string)
CurrentFlow (4)	(4 character string)
ID (8)	(8 character string)
Leftover (32)	

TOTAL SIZE = 240 + 80 = 320 Bytes (includes Header & Common Block)

3.5.5. DMC: RECON ARTILLERY

3.5.5.1. DMC: RECON ARTILLERY REPEAT

D C DMC PDU Specific:
05 Subtype = ARTY
00 Variation = REPEAT

Rev. Date: 11/23/92

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
fields are unsigned if not specified

Specific (56): Subtype.Repeat

3.5.5.2. DMC: RECON ARTILLERY CANCEL

DC DMC PDU Specific:
05 Subtype = ARTY
01 Variation = CANCEL

Rev. Date: 11/23/92

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
fields are unsigned if not specified

Specific (56): Subtype.Cancel

3.5.5.3. DMC: RECON ARTILLERY CHECK FIRE

DC DMC PDU Specific:
05 Subtype = ARTY
02 Variation = CHECK

Rev. Date: 11/23/92

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
fields are unsigned if not specified

Specific (56): Subtype.CheckFire

3.5.5.4. DMC: RECON ARTILLERY CNO

DC DMC PDU Specific:
05 Subtype = ARTY
03 Variation = CNO

Rev. Date: 11/23/92

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
fields are unsigned if not specified

Specific (56): Subtype.CNO

MissionID (16)	(16 character string)
TargetID (16)	(16 character string)
MissionStatus (1)	
Unused (7)	
Leftover (16)	

MissionStatus:
0 Not Entered
1 Requested
2 Ready
3 Shot
4 Splash

3.5.5.5. DMC: RECON ARTILLERY SHIFT

DC DMC PDU Specific:
05 Subtype = ARTY
04 Variation = SHIFT

Rev. Date: 11/23/92

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
fields are unsigned if not specified

Specific (56): Subtype.Shift

MissionID (16)	(16 character string)
TargetID (16)	(16 character string)
MissionStatus (1)	
FireforEffect (1)	
Unused (7)	
Message (16)	(16 character string)

MissionStatus:
0 Not Entered
1 Requested
2 Ready
3 Shot
4 Splash

FireforEffect:
: FALSE (OFF)
TRUE (ON)

3.5.5.6. DMC: RECON ARTILLERY NEW MISS

D C DMC PDU Specific:
05 Subtype = ARTY

05 Variation = NW_MSN

Rev. Date: 11/23/92

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
fields are unsigned if not specified

Specific (56): Subtype.NewMission

MissionID (16)	(16 character string)	
TargetID (16)	(16 character string)	
MissionStatus (1)		0 Not Entered 1 Requested 2 Ready 3 Shot 4 Splash
MissionType (1)		0 Not Entered 1 FFE 2 Adj Fire 3 Immed Suppr 4 Suppr
Shell (1)		0 Not Entered 1 HE 2 ICM 3 CprHd 4 Smoke 5 WP 6 Cont Illum
Control (1)		0 Not Ent. 1 AMC 2 WR 3 TOT
Fuze (1)		0 Not Entered 1 Quick 2 Time Delay 3 Concr Prcng
Trajectory (1)		0 Not Entered 1 Low 2 High 3 Other
RoundFFE (1)		
Unused (1)		
Leftover (16)		

3.5.5.7. DMC: RECON ARTILLERY END OF MISSION

D C DMC PDU Specific:
05 Subtype = ARTY

09 Variation = EOM

Rev. Date: 11/24/92

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
fields are unsigned if not specified

Specific (56): Subtype.EndofMission

MissionID (16)	(16 character string)	
TargetID (16)	(16 character string)	
MissionStatus (1)		0 Not Entered 1 Requested 2 Ready 3 Shot 4 Splash
Disposition (1)		0 Not Entered 1 Not Given 2 Burning 3 CND
Casualties (1)		0 Not Entered 1 Not Given 2 Given
RecordTarget (1)		0 FALSE 1 TRUE
CasualtyNumber (1)		
Unused (3)		
PointNumber (16)	(16 character string)	

TOTAL SIZE = 240 + 56 = 296 Bytes (includes Header & Common Block)

3.5.5.8. DMC: RECON ARTILLERY MTO

D C DMC PDU Specific:
05 Subtype = ARTY
06 Variation = MTO

Rev. Date:
11/24/92

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
fields are unsigned if not specified

Specific (56): Subtype.MTO

MissionID (16)	(16 character string)
TargetID (16)	(16 character string)
MissionStatus (1)	
Request Adjustment (1)	
EnterTarget (1)	
EndMission (1)	
Unused (4)	
Leftover (16)	

0 Not Entered
1 Requested
2 Ready
3 Shot
4 Splash

0 FALSE
1 TRUE

3.5.5.9. DMC: RECON ARTILLERY SHOT

D C DMC PDU Specific:
05 Subtype = ARTY
07 Variation = SHOT

Rev. Date:
11/24/92

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
fields are unsigned if not specified

Specific (56): Subtype.Shot

3.5.5.10. DMC: RECON ARTILLERY SPLASH

D C DMC PDU Specific:
05 Subtype = ARTY
08 Variation = SPLASH

Rev. Date:
11/24/92

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
fields are unsigned if not specified

Specific (56): Subtype.Splash

MissionID (16)	(16 character string)
TargetID (16)	(16 character string)
MissionStatus (1)	
RoundsFired (1)	
ImpactTime (1)	in seconds
Unused (5)	
Leftover (16)	

0 Not Entered
1 Requested
2 Ready
3 Shot
4 Splash

3.5.6. DMC: MOVI

D C DMC PDU Specific:
06 Subtype = MOVE
00 Variation = Standard

Rev. Date:
12/16/92

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
fields are unsigned if not specified

Specific (48):

TargetID (16)	(16 character string)
Task (1)	
Who (1)	
When (1)	
Unused (3)	
Zulu Time (2)	Minutes ahead of Zulu
Where (8)	(8 Character String)
DTG (16)	(See DTG Type)

Task (1):
0 Not Entered
1 TO
2 Hold
3 Cont Msn
4 Rendez At
5 Enga Tgt
6 Moving To
7 Hldg At
8 Arrvg At
9 Passg Thru
A Deprtg From

Who (1):
0 Not Entered
1 Mypos
2 B2
3 B3
4 Op1
5 Op2

When (1):
0 Not Entered
1 Immed
2 Whn Rdy
3 AMC
4 DTG

TOTAL SIZE = 240 + 48 = 288 Bytes (includes Header & Common Block)

3.5.7. DMC: STATUS

D C DMC PDU Specific:
07 Subtype = STATUS
00 Variation = Standard

Rev. Date:
11/23/92

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
fields are unsigned if not specified

Specific (16):

Fuel (1)	in lbs
Elements (1)	
FailedEquip (3)	(array of 3)
Hellfires (1)	
Stingers (1)	
Rockets (1)	
Rounds (1)	
RequestType (1)	
Unused (5)	

FailedEquip (3):
0 Not Entered
1 Eng1
2 Eng2
3 Guns
4 Radio
5 Dgntr
6 Rktpod
7 Mst Stn
8 Radar
9 Laser

RequestType (1):
0 Standard
1 Automatic

TOTAL SIZE = 240 + 16 = 256 Bytes (includes Header & Common Block)

3.5.8. DMC: REQUEST

D C DMC PDU Specific:
08 Subtype = REQUEST
00 Variation = Standard

Rev. Date: 11/23/92

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
fields are unsigned if not specified

Specific (8):

ReportType (1)	
ReconType (1)	
Unused (6)	

ReportType:
0 Not Entered
1 Status
2 Spot
3 Recon
4 Movement

ReconType:
0 Not Entered
1 Grnd Rte
2 Air Rte
3 Brdg
4 LZ/PZ
5 BP/OP
6 Crossing

TOTAL SIZE = 240 + 8 = 248 Bytes (includes Header & Common Block)

3.5.9. DMC: NBC

3.5.9.1. DMC: NBC-1

D C DMC PDU Specific:
09 Subtype = NBC
00 Variation = NBC-1

Rev. Date: 11/23/92

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
fields are unsigned if not specified

Specific (64): Subtype.NBC_1

Description (1)	
BurstType (1)	
DeliveredBy (1)	
CloudHtUnits (1)	
CloudWidth (4)	in degrees (32b float)
CloudDescr (16)	(16 character string)
FlashBangTime (2)	in seconds
Unused (8)	
StartDTG (16)	(See DTG Type)
EndDTG (16)	(See DTG Type)

Description:
0 Not Entered
1 Unusual
2 Initial
3 Increasing
4 Decreasing
5 Peak
6 Special
7 Sense
8 Verification
9 Summary

BurstType:
0 Not Entered
1 None
2 Unknown
3 Surface
4 Air
5 Gnd

DeliveredBy:
0 Not Entered
1 Unknown
2 Arty
3 Mortar
4 Rocket
5 Missile
6 Bomb

CloudHtUnits:
0 Not Entered
1 Degrees
2 Ht in Ft
3 Ht in M

3.5.9.2. DMC: NBC-4

D C DMC PDU Specific:
09 Subtype = NBC
03 Variation = NBC-4

Rev. Date: 11/23/92

LEGEND
B = Bytes (default if not specified)
b = bits
all values specified in hex
fields are unsigned if not specified

Specific (64): Subtype.NBC_2

Description (1)	
BurstType (1)	
DeliveredBy (1)	
CloudHtUnits (1)	
CloudWidth (4)	in degrees (32b float)
CloudDescr (16)	(16 character string)
DoseRate (2)	
Unused (6)	
Leftover (32)	

Description:
0 Not Entered
1 Unusual
2 Initial
3 Increasing
4 Decreasing
5 Peak
6 Special
7 Sense
8 Verification
9 Summary

BurstType:
0 Not Entered
1 None
2 Unknown
3 Surface
4 Air
5 Gnd

DeliveredBy:
0 Not Entered
1 Unknown
2 Arty
3 Mortar
4 Rocket
5 Missile
6 Bomb

CloudHtUnits:
0 Not Entered
1 Degrees
2 Ht in Ft
3 Ht in M

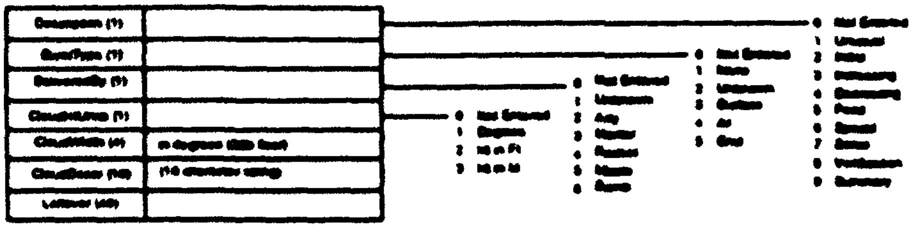
3.5.9.3. DMC: NBC-5

D C DMC PDU Specific:
09 Subtype = NBC

04 Variation = NBC-5

Rev. Date:
11/23/92

Specific (64): Subtype.NBC_Negative



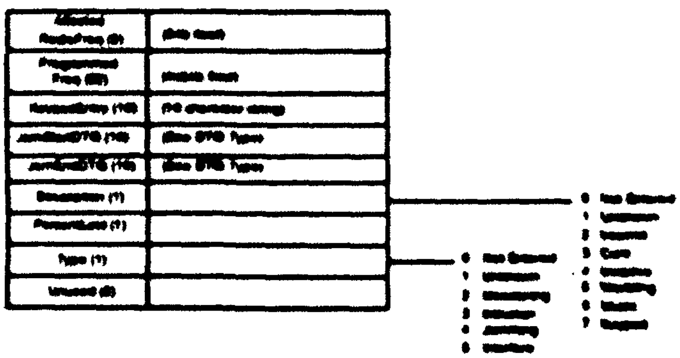
TOTAL SIZE = 240 + 64 = 304 Bytes (includes Header & Common Block)

3.5.10. DMC: MIJI

D C DMC PDU Specific:
0 A Subtype = MIJI
00 Variation = Standard

Rev. Date:
11/23/92

Specific (96):



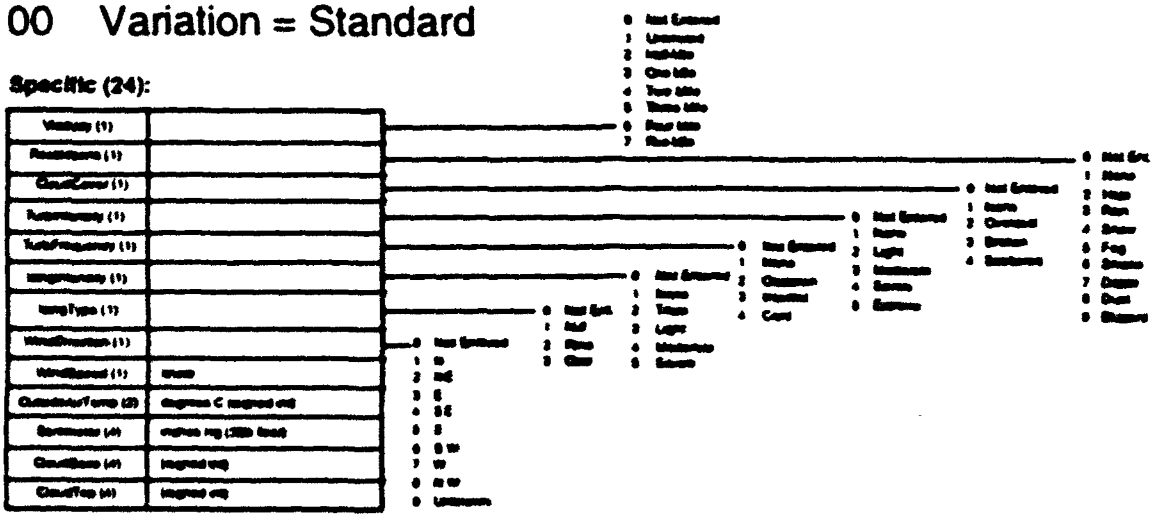
TOTAL SIZE = 240 + 96 = 336 Bytes (includes Header & Common Block)

3.5.11. DMC: PIREP

D C DMC PDU Specific:
0 B Subtype = PIREP
00 Variation = Standard

Rev. Date: 11/23/92

Specific (24):



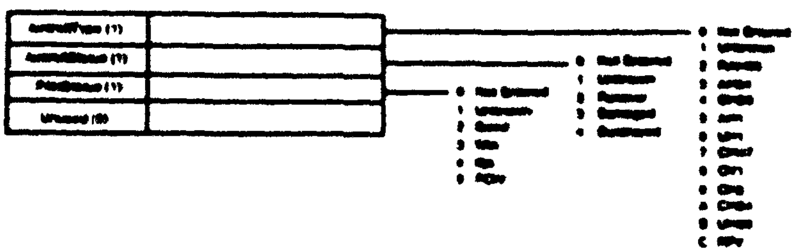
TOTAL SIZE = 240 + 24 = 264 Bytes (includes Header & Common Block)

3.5.12. DMC: DNAV

D C DMC PDU Specific:
0 C Subtype = DNAV
00 Variation = Standard

Rev. Date: 11/23/92

Specific (8):



TOTAL SIZE = 240 + 8 = 248 Bytes (includes Header & Common Block)

3.5.13. DMC: FREE TEXT

D C DMC PDU Specific:
0 D Subtype = FREE TEXT
00 Variation = Standard

Rev. Date:
11/23/92

LEGEND
[illegible]

Specific (256):

Free Text(256)	(256 character string)

TOTAL SIZE = 240 + 256 = 496 Bytes (includes Header & Common Block)

4. ENUMERATED AND BIT ENCODED VALUES FOR USE WITH DIGITAL MESSAGE/COMMUNICATIONS PROTOCOL (DMCP)

4.1. UPDATED FIELDS

See specifications in Section 3.

4.2. NEW FIELDS

See specifications in Section 3.

APPENDIX C: SIMNET PROTOCOL EXTENSIONS

Component Protocol Extensions

1 Smart Mines Simulation Protocol Extension
2 Data Collection Protocol Extension
3 Missile Server Protocol Extension
4 CVCC Protocol Extension
5 MultiRad Protocol Extension
6 Persistent Object Protocol
7 VIDS Protocol Extension

APPENDIX C1: SMART MINES SIMULATION PROTOCOL EXTENSION

1. Smart Minefield Simulator (SMS) Protocol Extension

This appendix describes the protocol extension developed to support of the Smart Minefield Simulator (SMS) Version 1.6. The SMS simulator models four kinds of mines, including the generic conventional anti-tank mine, the Textron anti-tank wide-area mine (WAM), the Textron anti-helicopter mine (AHMT), and the Ferranti anti-helicopter mine (AHM-F).

The SMS runs on a Masscomp computer at a 10 Hz frame rate. Most user-interface functions are supported on a PC-clone, which communicates with the SMS via the SMS protocol extension.

1.1. Base Standard

The SMS protocol is an extension to the SIMNET 6.6.1 protocol as documented in [BBN 1991]. The primary purpose of the protocol is to allow the SM user interface to communicate with the SM simulator. No other simulation applications issue or respond to the SMS protocol. The SMS communicates with other simulations using the core SIMNET protocol.

The SMS protocol is a test specific protocol and no plans have been made to integrate this protocol into the base SIMNET standard.

1.2. Applicable documents.

The following documents are referenced in this protocol extension:

BBN 1991 Pope, Arthur R.; Schaffer, Richard L. SIMNET Networks and Protocols, BBN Systems and Technologies, BBN-7627, June 1991.

1.3. Implementation History.

The Smart Minefield Simulation protocol was developed to support the SMS study. This ADST delivery order developed the software to replicate a smart minefield according to functional specifications developed by IDA (in consultation with Loral); the software supported the replication of the smart minefield and associated weapons was be hosted on existing BDS-D hardware. This was a small scale effort (follow-on to WAMS) to implement the smart mines on the BDS-D battlefield.

The SMS study conducted tests using Smart Mines Simulator and Semi-Automated Forces only. The SM Simulator was developed for the SMS study and is not currently in wide spread use.

2. General Requirements

2.1. Introduction

The SMS protocol extensions purpose is to allow the SM user interface to communicate with the SM simulator. No other simulation applications issue or

respond to the SMS protocol. The SMS protocol, as recorded by the data logger, is also used for After Action Review (AAR) and analysis purposes.

2.1.1. Terminology

This paragraph is intentionally left blank.

2.1.2. Key Concepts

This paragraph is intentionally left blank.

2.1.3. Information common to all PDUs in this extension.

The SMS protocol is implemented as a SIMNET sub-protocol. The SMS protocol has been assigned a protocol number and protocol version so that it can be discriminated from other sub-protocols.

2.2. PDUs for SMS Protocol Extension

2.2.1. SMS Emplacement PDU.

The SMS Emplacement PDU shall be used to communicate the emplacement of a mine field.

2.2.1.1. Information Contained in the SMS Emplacement PDU

The SMS Emplacement PDU contains the following information:

(1) PDU Header

(2) Emplacement method of minefield

(3) Number of points necessary to define the emplacement.

(4) Number of mines to be laid in the mine field.

(5) Width, in meters, of the minefield.

(6) The type of mine to be emplaced.

(7) Density, in mines/km**2, of emplaced mines (alternative to quantity specification)

(8) Mine field identifier.

(9) UTM locations as prescribed by emplacement method and number of points in line.

2.2.1.2. Issuance of the SMS Emplacement PDU

The SMS emplacement PDU is issued by the SM user interface application to create a minefield with the given parameters.

2.2.1.3. Receipt of the SMS Emplacement PDU

Upon receipt of the SMS Emplacement PDU the SM Simulator will create a minefield with the given parameters.

2.2.2. SMS Control PDU.

The SMS Control PDU shall be used to communicate changes in state of minefields.

2.2.2.1. Information Contained in the SMS Control PDU

The SMS Control PDU contains the following information:

(1) PDU Header

(2) Control Type specifies what target SMS component is to be controlled.

(3) Index specifies the identifier of the target SMS component.

(4) Value specifies the new state of the target SMS component.

2.2.2.2. Issuance of the SMS Control PDU

The SMS Control PDU is issued by the SM user interface to change the state of a minefield on the SMS.

2.2.2.3. Receipt of the SMS Control PDU

Upon receipt of the SMS Control PDU the SMS will change the state of the indicated minefield according to the PDU parameters.

2.2.3. SMS Status PDU.

The SMS Status PDU shall be used to communicate the status of the SMS to the SM user interface.

2.2.3.1. Information Contained in the SMS Status PDU

The SMS Status PDU contains the following information:

(1) PDU Header

(2) Vehicle ID

(3) Status Type

(4) Field Status, or

  a) Field identifier

  b) Field state

  c) Field location

  d) Quantity of mines in field

(4) Sensor Status

  a) Sensor state

  b) Field identifier

  c) Sensor identifier

  d) Sensor type

e) Sensor location

2.2.3.2. Issuance of the SMS Status PDU

A SMS Status PDU describing a field is issued whenever the contents of the field change, whenever the field's state is changed by the user, and every 60 seconds. A SMS Status PDU describing a sensor is issued whenever the sensor's state changes.

2.2.3.3. Receipt of the SMS Status PDU

Upon receipt of the SMS Status PDU the SM user interface will update its world view.

3. Detailed Requirements

3.1. Introduction

This section defines the PDUs and their fields.

3.2. Representation of Data

This protocol extension does not introduce any new types of data representation.

3.3. Basic Data Types and Records

3.3.1. PDU Header

The PDU header shall be the first part of each SMS protocol PDU. This header contains the following fields:

1) protocol version number
2) protocol data unit type
3) exercise identification.

See SIMNET Network and Protocols for a complete description of the PDU Header. [BBN 1991]

3.4. List of PDUs in Protocol Extension

This SMS protocol extension is comprised of the following PDUs:

1) SMS Emplacement PDU
2) SMS Control PDU.
3) SMS Status PDU.

3.5. Protocol Data Units for the SMS Protocol Extensions

3.5.1. SMS Emplacement PDU Information

The emplacement of a minefield shall be communicated by issuing a SMS Emplacement PDU. The SMS Emplacement PDU shall contain the following fields:

(1) PDU Header - See 3.3.1.

(2) Emplacement Method - This field shall describe how the minefield should be emplaced. The emplacement method determines the number of UTMs required to define the minefield and if a width is required. See 4.2.1.

(3) Points in line - If the Emplacement Method is "InLine" or "InArea" then this field is required and specifies the number of UTMs that are needed to define this emplacement. This field cannot be greater than 10 and will be set to 0 if not required.

(4) Quantity - This field will contain the number of mines the mine field is to contain.

(5) Width - If the Emplacement Method is "InLine" then this field is the width, in meters, of the minefield otherwise this field is set to 0.

(6) Mine Type - This field shall describe the type of mines the mine field should contain. See 4.2.2.

(7) Density - An alternative method of defining the number of mines to be emplaced is to specify the desired density in mines per square kilometer. This field is used if the quantity is zero.

(8) Field - This field shall uniquely identify the mine field to be emplaced.

(9) UTM - This field shall specify a UTM locations used to define the location of the mine field. The number of locations is dependent on the emplacement method, and possibly on the number of points in line. For individual and rectangle, it is one and two, respectively. For line and area, it is the points_in_line value.

The SMS Emplacement PDU is represented in Fig 3-1.

Field Size (bits)	SMS Emplacement PDU	
64	PDU Header	Protocol Version 8 bit char
		PDU Kind 8 bit char
		Exercise 8 bit char
		Padding 40 bits - Unused
8	Emplacement Method *	8 bit enumerated
8	Points in line	8 bit integer
16	Quantity	16 bit integer
16	Width	16 bit integer
16	Mine Type	16 bit enumerated
32	Density	32 bit integer
32	Field	32 bit integer
Varies n x 128	UTM(s)	array of 16 char

Figure 3-1. SMS Emplacement PDU.

3.5.2. SMS Control PDU Information

The Control of a minefield sh 'l be communicated by issuing a SMS Control PDU. The SMS Control PDU sh . contain the following fields:

(1) PDU Header - See 3.3.1.

(2) Control Type - This field specifies what target SMS component is to be controlled. See 4.2.3.

(3) Index - This field specifies the identifier of the target SMS component.

(4) Value - This field specifies the new state of the target SMS component. See 4.2.4.

The SMS Control PDU is represented in Fig 3-2.

Field Size (bits)	SMS Control PDU	
64	PDU Header	Protocol Version 8 bit char
		PDU Kind 8 bit char
		Exercise 8 bit char
		Padding 40 bits - Unused
	Control Type	32 bit enumerated
8	Index	32 bit integer
16	Value	32 bit enumerated

Figure 3-2. SMS Control PDU.

3.5.3. SMS Status PDU Information

The SMS Status PDU shall be used to communicate the status of the SMS mine fields and sensors to the SM user interface. The SMS Status PDU shall contain the following fields:

(1) PDU Header - See 3.3.1.

(2) Vehicle ID - This field contains the vehicle id of the SMS simulation.

(3) Status Type - This field specifies the kind of object being described. See 4.2.3.

(4) Field Status or Sensor Status - If the PDU is reporting on the status of a mine field the following fields will be required:

a) Field - This field is the unique identifier for the mine field.

b) State - This field describes the current status of the mine field. See 4.2.5.

c) Lower Left UTM - This field defines the location of a point south and west of all mines in the field.

d) Upper Right UTM - This field defines a point north and east of all mines in the field.

e) Quantity[NumMineTypes] - This field specifies the number of each kind of mine in the field.

If the PDU is reporting on the status of a sensor the following fields will be required:

a) State - This field describes the current state of the sensor. See 4.2.6.

b) Field - This field is the field identifier of the sensor's minefield.

c) Index - This field is the unique identifier for the sensor.

d) Type - This field is the type of sensor. See 4.2.7

e) Location - This field is the location of the sensor on the battlefield in world coordinates.

The SMS Status PDU is represented in Fig 3-3.

<table>
<tr><th>Field Size (bits)</th><th colspan="3">SMS Status PDU</th></tr>
<tr><td rowspan="4">64</td><td colspan="2" rowspan="4">PDU Header</td><td>Protocol Version 8 bit char</td></tr>
<tr><td>PDU Kind 8 bit char</td></tr>
<tr><td>Exercise 8 bit char</td></tr>
<tr><td>Padding 40 bits - Unused</td></tr>
<tr><td>16</td><td colspan="2">VehicleID</td><td>16 bit integer</td></tr>
<tr><td>16</td><td colspan="2">Status Type</td><td>16 bit integer</td></tr>
<tr><td rowspan="11">if Field Status 304 + n x 16

if Sensor Status 160</td><td rowspan="11">Field Status

OR

Sensor Status</td><td rowspan="5">Field Status</td><td>Field 32 bit integer</td></tr>
<tr><td>State 16 bit integer</td></tr>
<tr><td>Lower Left UTM 16 - 8 bit char</td></tr>
<tr><td>Upper Right UTM 16 - 8 bit char</td></tr>
<tr><td>Quantity[n] 16 bit integer</td></tr>
<tr><td rowspan="6">Sensor Status</td><td>State 8 bit char</td></tr>
<tr><td>Field 8 bit char</td></tr>
<tr><td>Index 16 bit integer</td></tr>
<tr><td>Type 8 bit char</td></tr>
<tr><td>padding 24 bit unused</td></tr>
<tr><td>Location 3 - 32 bit Integers</td></tr>
</table>

Figure 3-3. SMS Status PDU.

4. Enumerated and Bit Encoded Values for Use with the SMS Protocol Extension

The material for in this section should be suitable for insertion in the "Enumerated and Bit Encoded Values for Use with Protocols for Distributed Interactive Simulation Applications" document that accompanies the base standard (see section 1.1). If this is a SIMNET protocol extension, the information should still be in this format.

4.1. Updated Fields

4.1.1. Protocol Number

To distinguish the SMS protocol from other protocols using the association sublayer, the SMS protocol is assigned a unique association sublayer user protocol number [BBN 1991, p80]. This number is 149. Protocol Number is discussed Section 7.2 of [BBN 1991]

Field Value	Protocol Version	Meaning
149	smsProtocol Number	Current protocol number

4.1.2. Protocol Version

The following protocol version values have been defined for the SMS protocol.

Field Value	Protocol Version	Meaning
1	smsProtocolVersionSep92	Current protocol Version

4.1.3. PDU Kind

The following values are defined for this field by the protocol extension.

Field Value	PDU Kind
1	SMS Emplacement PDU
2	SMS Control PDU
3	SMS Status PDU

4.2. New Fields

4.2.1. Emplacement Method

This field is referenced by the SMS Emplacement PDU. See 3.5.1.

Field Value	Emplacement Method	Meaning
0	Individual	Single Mine - one UTM is defined
1	InRectangle	Emplace mines in a rectangle area defined by two UTMs
2	InLine	Emplace mines along a line defined by up to 10 UTMs are defined plus a width
3	InArea	Emplace mines in an area defined by up to 10 UTMs

4.2.2. Mine Type

This field is referenced by the SMS Emplacement PDU. See 3.5.1.

Field Value	Mine Type	Meaning
1	Conventional	generic anti-tank mine
2	WAM	Textron anti-tank wide-area mine
3	AHM-T	Textron anti-helicopter mine
4	AHM-F	Ferranti anti-helicopter mine

4.2.3. Control/Status Type

This field is referenced by the SMS Control PDU, see 3.5.2, and the SMS Status PDU, see 3.5.3.

Field Value	Control Type	Meaning
1	Field	NA
2	Mine	NA
3	Debug	NA

4.2.4. Value

This field is referenced by the SMS Control PDU. See 3.5.2.

Field Value	Value	Meaning
0	Off	NA
1	On	NA
2	Detonate	NA
3	Clear	NA

4.2.5. Mine Field State

This field is referenced by the SMS Status PDU, mine field variant. See 3.5.3.

Field Value	Status Type	Meaning
0	Off	Object is off
1	On	Object is on
2	Track	Mine Field is in tracking state
3	Detect	Mine Field is in detection state

4.2.6. Sensor State

This field is referenced by the SMS Status PDU, sensor variant. See 3.5.3.

Field Value	Status Type	Meaning
0	Off	Object is off
1	On	Object is on
2	Close	Sensor is in close state
3	Tracking	Sensor is in tracking state
4	Detection	Sensor is in detection state

4.2.7. Sensor Type

This field is referenced by the SMS Status PDU, sensor variant. See 3.5.3.

Field Value	Status Type	Meaning
0	Proximity Sensor	NA
1	WAM Sensor	NA
2	AHMF Sensor	NA
3	AHMT Sensor	NA

DATA COLLECTION PROTOCOL EXTENSIONS

Hollis Experiment

In support of the Hollis Experiment, enhancement were made to the data collection protocol. These changes allowed the simulators to more accurately report on the acquisition process.

LOSAT

For the Line of Sight Anti Tank simulator the following experiment specific PDUs were add to the Data Collection Protocol.

- SP_LosatStatusVariant
- SP_LosatAidedCueingVariant
- SP_LosatAidedScanVariant
- SP_LosatAutoTrackVariant
- SP_LosatRangeVariant

Appendix C2:
Data Collection Protocol Extensions

APPENDIX C3: MISSILE SERVER PROTOCOL

MISSILE SERVER PROTOCOL.

In support of the Air to Air Combat II (ATAC II) Delivery Order, a Missile Server was added to the network to allow firing Hellfire missiles with a remote designator. Prior to this extension, missile flyout was limited by the 7 km range limitation on RWA devices. The Missile Server handles missile flyout and enhances intervisibility calculations. These enhancement implemented the Missile Server Protocol.

The following figure is a notional representation of the PDUs that were developed and how they are used by the hosts.

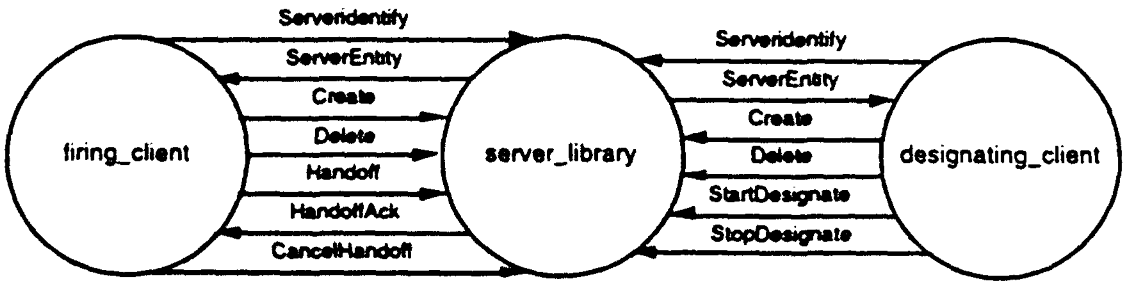


Figure C3-1. ATAC II Missile Server Protocol.

APPENDIX C4: CVCC PROTOCOL EXTENSION

APPENDIX C5: MULTIRAD PROTOCOL EXTENSIONS

This appendix contains the attachment "SIMNET 6.6.1+ Network Protocols For The TRUE and WARBREAKER Programs, Appendix C5, Attachment 1."

SIMNET 6.6.1+ NETWORK PROTOCOLS FOR THE TRUE & WAR BREAKER PROGRAMS

1 December 1992

Document No. AL0692-009 Rev. E

Prepared for:
Air Force Human Resources Laboratory
Williams Air Force Base, AZ

Prepared by:

LORAL

SIMNET 6.6.1+
NETWORK PROTOCOLS
FOR THE
TRUE & WAR BREAKER
PROGRAMS

REVISION HISTORY

REVISION	DATE	COMMENT
Rev. N/C	2 April 1992	Update
Rev. A	14 April 1992	Update
Rev. B	22 June 1992	Update
Rev. C	8 September 1992	Added Appendix B & BBN Guises to Appendix A
Rev. D	22 October 1992	Added SA-2 & SA-3 Missile Guises to Appendix A, page A-8
Rev. E	1 December 1992	Revised Radar & Emitter PDUs on Pages 16 - 20

SIMNET 6.6.1+ NETWORK PROTOCOLS FOR THE TRUE & WAR BREAKER PROGRAMS

TABLE OF CONTENTS

1.0 Introduction

This paper identifies the protocols which will be used for the TRUE and WAR BREAKER Programs. It includes both SIMNET 6.6.1 protocols and extensions to them.

2.0 Protocol Data Units

2.1 Activate Request PDU

One network device may prompt another to begin simulating a vehicle through a activate request. The Activate Request PDU includes the following data:

FIELD SIZE (bits)	ACTIVATE REQUEST PDU FIELDS	
8	PROTOCOL VERSION	8-bit unsigned integer
8	PDU TYPE	8-bit unsigned integer
8	EXERCISE ID	8-bit unsigned integer
40	PADDING	40-bit unsigned integer
8	ACTIVATE REASON	8-bit unsigned integer
8	VEHICLE CLASS	8-bit unsigned integer
48	VEHICLE ID	Site - 16-bit unsigned integer
		Host - 16-bit unsigned integer
		Vehicle - 16-bit unsigned integer
160	ORGANIZATIONAL UNIT	Force ID - 8-bit unsigned integer
		Organization Type - 8-bit unsigned integer
		Unit Identifier - 18 - 8-bit unsigned integers
96	MARKING	Character Set - 8-bit integer
		Text - 11 - 8-bit characters
64	VEHICLE GUISES	Distinguished - 32-bit unsigned integer
		Other - 32-bit unsigned integer
32	SIMULATED TIME	32-bit unsigned integer

FIELD SIZE (bits)	ACTIVATE REQUEST PDU CONTINUED			
128	TERRAIN DATABASE ID	Terrain Name - 14 - 8-bit characters		
		Terrain Version - 16-bit unsigned integer		
8	BATTLE SCHEME	8-bit unsigned integer		
1	ON SURFACE	1-bit unsigned integer		
23	PADDING	23-bit integer		
960	VEHICLE STATUS	Vehicle Type - 32-bit unsigned integer		
		Odometer - 32-bit floating point		
		Age - 8-bit unsigned integer		
		Unused - 24-bits		
		Failures (Vehicle Subsystems) - 416-bits		
		Status Category - 16-bit unsigned integer		
		Padding - 16-bit integer		
		Engine Power - 8-bit unsigned integer	Generic Status Category (A/C)	B
		Battery Voltage - 24-bit unsigned integer		
		Munition Record [6]: Type - 32-bit unsigned integer		
		Munition Record [6]: Quantity - 32-bit floating point		
192	LOCATION (WORLD COORDINATES)	x - 64-bit floating point		
		y - 64-bit floating point		
		z - 64-bit floating point		
64	SIMPLE VEHICLE DATA (A/C)	Yaw - 32-bit BAM		
		Padding - 32-bit integer		
96	VELOCITY	x - 32-bit floating point		B
		y - 32-bit floating point		
		z - 32-bit floating point		
1	FREEZE STATE	1-bit unsigned integer		
31	PADDING	31-bit unsigned integer		
32	VLVIS	32-bit floating point		

FIELD SIZE (bits)	ACTIVATE REQUEST PDU CONTINUED	
8	SKY COLOR	8 - bit unsigned integer
24	PADDING	24 - bit integer
32	FUEL QUANTITY	32-bit floating point
16	RADIO CHANNEL	16-bit unsigned integer
16	MISSION #	16-bit unsigned integer
1536	WAYPOINTS [16]	Lat - 32-bit floating point
		Lon - 32-bit floating point
		Alt - 32-bit floating point

B

Total Activate Request PDU Size = 3648 bits B

Simulation PDU header information
- PROTOCOL VERSION — SIMNET protocol version used in the variant portion of the PDU
- PDU TYPE — PDU type to follow in the variant portion of the packet
- EXERCISE ID — Exercise generating PDU (important when multiple exercises on network)

Activate Request Variant
- ACTIVATE REASON — Reason to activate the vehicle
  - 0 Activate reason other
  - 1 Exercise start
  - 2 Exercise restart
  - 3 Vehicle reconstitution
  - 4 Towing arrival
- VEHICLE CLASS — Class for number of independently moveable parts for RVA
  - 0 Vehicle class irrelevant
  - 1 Vehicle class static
  - 2 Vehicle class simple
  - 3 Vehicle class tank
- VEHICLE ID — Vehicle identification
  - Simulation address — Site
    - Host
  - Vehicle
- ORGANIZATIONAL UNIT — Organizational hierarchy (not currently used)
- MARKING — Character string of vehicle markings
- VEHICLE GUISES
  - Distinguished — As seen by blue team

Other		As seen by other teams
Bit field		
	Domain	3
	Environment	3
	Class	3
	Class	3
	Country	6
	Series	6
	Model	6
	Function	5
SIMULATED TIME		Time being simulated
TERRAIN DATABASE ID		Database being used
BATTLE SCHEME		Identifies how force ID's and guises are being used
0	Battle scheme other	
1	Battle scheme absolute (does not use guises)	
2	Battle scheme relative (uses guises)	
ON SURFACE		Indicates if vehicle is on the surface of the database or in flight
VEHICLE STATUS		Contains status of vehicle. The only field currently used is munitions.
LOCATION		Location in world coordinates (meters)
VEHICLE DATA - YAW		Initial rotation of vehicle (BAM)
VELOCITY		Initial velocity (meters per second)
FREEZE STATE		Initial freeze mode
0	Unfreeze	
1	Freeze	
VLSVIS		Visibility in visible light (meters)
SKY COLOR		Simulated sky color
FUEL QUANTITY		Initial fuel (pounds)
RADIO CHANNEL		Radio channel
MISSION NUMBER		Number of mission for initialization
WAYPOINTS		Lat, lon and alt of 16 waypoints

2.2 Activate Response PDU

A network device that correctly receives an Activate Request must immediately respond by returning an Activate Response. The Activate Response includes the following data:

FIELD SIZE (bits)	ACTIVATE RESPONSE PDU FIELDS	
8	PROTOCOL VERSION	8-bit unsigned integer
8	PDU TYPE	8-bit unsigned integer
8	EXERCISE ID	8-bit unsigned integer
40	PADDING	40-bit unsigned integer
48	VEHICLE ID	Site - 16-bit unsigned integer
		Host - 16-bit unsigned integer
		Vehicle - 16-bit unsigned integer
8	RESULT	8-bit unsigned integer
8	PADDING	8-bit unsigned integer
16	TIME LIMIT	16-bit unsigned integer
16	PADDING	16-bit integer
32	PADDING	32-bit integer

Total Activate Response PDU Size = 192 bits

Simulation PDU header information

- PROTOCOL VERSION — SIMNET protocol version used in the variant portion of th PDU
- PDU TYPE — PDU type to follow in the variant portion of the packet
- EXERCISE ID — Exercise generating PDU (important when multiple exercises on network)

Activate response variant

- VEHICLE ID — Vehicle identification
  - Simulation address
    - Site
    - Host
  - Vehicle
- REASON
  - 0 Activate request accepted
  - 1 Invalid activation parameter
  - 2 Unexpected activate reason
  - 3 Invalid vehicle identifier
  - 4 Terrain database unavailable
- TIME LIMIT — Not currently used

2.3 Deactivate Request PDU

A network device may withdraw its own vehicles from an exercise at any time, or it may be requested by another simulator to withdraw. In either case, the withdrawal of the vehicle is announced using a Deactivation.

FIELD SIZE (bits)	DEACTIVATE REQUEST PDU FIELDS	
8	PROTOCOL VERSION	8-bit unsigned integer
8	PDU TYPE	8-bit unsigned integer
8	EXERCISE ID	8-bit unsigned integer
40	PADDING	40-bit unsigned integer
48	VEHICLE ID	Site - 16-bit unsigned integer
		Host - 16-bit unsigned integer
		Vehicle - 16-bit unsigned integer
8	REASON	8-bit unsigned integer
8	PADDING	8-bit unsigned integer
32	TIME STAMP	32-bit unsigned integer

B

Total Deactivate Request PDU Size = 160 bits B

Simulation PDU header information

- PROTOCOL VERSION — SIMNET protocol version used in the variant portion of the PDU
- PDU TYPE — PDU type to follow in the variant portion of the packet
- EXERCISE ID — Exercise generating PDU (important when multiple exercises on network)

Deactivate request variant

- VEHICLE ID — Vehicle identification
  - Simulation address — Site
    - Host
  - Vehicle
- REASON — Reason for deactivation
  - 0 Deactivate reason other
  - 1 Exercise end

2 Vehicle withdrawn
3 Vehicle destroyed
4 Towing departure

TIME STAMP Time of PDU issuance

2.4 Vehicle Appearance PDU

A simulator/network device periodically reports information about a vehicle it simulates s that other devices on the network may depict that vehicle. A network device will issue a ne Vehicle Appearance for a vehicle whenever the discrepancy between the vehicle's actu appearance and its dead reckoned appearance exceeds one of the defined thresholds. will also issue a new Vehicle Appearance if 5 seconds have elapsed since its la transmittal. A Vehicle Appearance includes the following data:

FIELD SIZE (bits)	VEHICLE APPEARANCE PDU FIELDS	
8	PROTOCOL VERSION	8-bit unsigned integer
8	PDU TYPE	8-bit unsigned integer
8	EXERCISE ID	8-bit unsigned integer
40	PADDING	40-bit unsigned integer
48	VEHICLE ID	Site - 16-bit unsigned integer
		Host - 16-bit unsigned integer
		Vehicle - 16-bit unsigned integer
8	VEHICLE CLASS	8-bit unsigned integer
8	FORCE ID	8-bit unsigned integer
64	VEHICLE GUISES	Distinguished - 32-bit unsigned integer
		Other - 32-bit unsigned integer

B

FIELD SIZE (bits)	VEHICLE APPEARANCE PDU CONTINUED		
192	LOCATION (WORLD COORDINATES)	x - 64-bit floating point	
		y - 64-bit floating point	
		z - 64-bit floating point	
288	ROTATION MATRIX	9 - 32-bit floating points	
32	APPEARANCE	32-bit unsigned integer	
96	MARKING	Character Set - 8-bit integer	
		Text - 11 - 8-bit characters	
32	TIME STAMP	32-bit unsigned integer	
32	CAPABILITIES	32-bit unsigned integer	
16	ENGINE SPEED	16-bit unsigned integer	
1	STATIONARY	1-bit unsigned integer	
7	PADDING	7-bit integer	B
8	REASON	8-bit unsigned integer	B
96	LINEAR VELOCITY VECTOR	x - 32-bit floating point	Vehicle Class Simple
		y - 32-bit floating point	Vehicle Class Simple
		z - 32-bit floating point	Vehicle Class Simple
32	PADDING	32-bit unsigned integer	Vehicle Class Simple
96	LINEAR ACCEL. VECTOR	x - 32-bit floating point	Vehicle Class Simple
		y - 32-bit floating point	Vehicle Class Simple
		z - 32-bit floating point	Vehicle Class Simple
96	ANGULAR VELOCITY VECTOR	pitch rate - 32-bit floating point	Vehicle Class Simple
		roll rate - 32-bit floating point	Vehicle Class Simple
		yaw rate - 32-bit floating point	Vehicle Class Simple
32	THROTTLE POSITION	32-bit floating point	
32	FUEL QUANTITY	32-bit floating point	

Total Vehicle Appearance PDU Size = 1280 bits B

Simulation PDU header information

PROTOCOL VERSION — SIMNET protocol version used in the variant portion of the PDU
PDU TYPE — PDU type to follow in the variant portion of the packet
EXERCISE ID — Exercise generating PDU (important when multiple exercises on network)

Vehicle Appearance variant

- VEHICLE ID — Vehicle identification
  - Simulation address
    - Site
    - Host
  - Vehicle
- VEHICLE CLASS — Class for number of independently moveable parts for RVA
  - 0 Vehicle class irrelevant
  - 1 Vehicle class static
  - 2 Vehicle class simple
  - 3 Vehicle class tank
- FORCE ID — Force identifier
  - 0 Force ID irrelevant
  - 1 Distinguished force ID
  - 2 Other force ID
  - 3 Observer force ID
  - 4 Target force ID
- VEHICLE GUISES
  - Distinguished — As seen by blue team
  - Other — As seen by other teams
  - Bit field
    - Domain 3
    - Environment 3
    - Class 3
    - Country 6
    - Series 6
    - Model 6
    - Function 5
- LOCATION — Location in world coordinates (meters)
- ROTATION MATRIX — 3x3 rotation matrix for vehicle orientation
- APPEARANCE — Bit field

BIT	PURPOSE
0	Vehicle destroyed (1=true)
1	Vehicle smoke plume (1=true)
2	Vehicle flaming (1=true)
3-4	Vehicle dust cloud 0 No dust cloud 1 Small dust cloud 2 Medium dust cloud 3 Large dust cloud
5	Vehicle mobility disabled (1=true)
6	Vehicle fire power disabled

7	Vehicle communications disabled	
8	Vehicle shaded (1=vehicle in shadow)	
30	Vehicle TOW launcher up	
31	Vehicle engine smoke	

MARKING	Character string of vehicle markings
TIME STAMP	Time PDU was issued
CAPABILITIES	Capabilities of the vehicle (bit field)
ENGINE SPEED	Engine speed (Revolutions per second)
STATIONARY	Flag variable
REASON	Reason for issuing PDU
LINEAR VELOCITY VECTOR	Velocity vector in world coordinates (m/s)
LINEAR ACCELERATION	Acceleration vector (m/s2)
ANGULAR VELOCITY	Angular velocity vector (rad/s)
THROTTLE POSITION	Engine throttle position
FUEL QUANTITY	Pounds of fuel remaining

2.5 Fire PDU

A Fire describes the firing of a shell, a burst of machine gun fire, or a missile. It is issued by the firing vehicle simulator.

FIELD SIZE (bits)	FIRE PDU FIELDS	
8	PROTOCOL VERSION	8-bit unsigned integer
8	PDU TYPE	8-bit unsigned integer
8	EXERCISE ID	8-bit unsigned integer
40	PADDING	40-bit unsigned integer
48	ATTACKER ID	Site - 16-bit unsigned integer
		Host - 16-bit unsigned integer
		Vehicle - 16-bit unsigned integer
16	EVENT ID	16-bit unsigned integer

8

FIELD SIZE (bits)	FIRE PDU CONTINUED		
96	BURST DESCRIPTOR	Projectile - 32-bit unsigned integer	
		Detonator - 32-bit unsigned integer	
		Quantity - 16-bit unsigned integer	
		Rate - 16-bit unsigned integer	
64	TARGET DESCRIPTOR	Target Type - 8-bit integer	
		Unused - 8-bit integer	
		Site - 16-bit unsigned integer	
		Host - 16-bit unsigned integer	
		Vehicle - 16-bit unsigned integer	
96	VELOCITY VECTOR	x - 32-bit floating point	
		y - 32-bit floating point	
		z - 32-bit floating point	
192	LOCATION (WORLD COORDINATES)	x - 64-bit floating point	
		y - 64-bit floating point	
		z - 64-bit floating point	
48	PROJECTILE ID	Site - 16-bit unsigned integer	
		Host - 16-bit unsigned integer	
		Vehicle - 16-bit unsigned integer	
8	PADDING	8-bit unsigned integer	
8	FIRE TYPE	8-bit unsigned integer	
128	SHELL FIRE DESCRIPTOR	Range - 32-bit floating point	FIRE TYPE = shell
		Slew Rate - 32-bit floating point	
		Ammo Type - 32-bit unsigned integer	
		Padding - 32-bit integer	
	MISSILE FIRE DESCRIPTOR	Tube - 8-bit unsigned integer	FIRE TYPE = missile
		Padding - 8-bit unsigned integer	
		Padding - 16-bit integer	
		Padding - 32-bit integer	
		Padding - 32-bit integer	
		Padding - 32-bit integer	

FIELD SIZE (bits)	FIRE PDU CONTINUED	
32	TIME STAMP	32-bit unsigned integer

Total Fire PDU Size = 800 bits | B

Simulation PDU header information

PROTOCOL VERSION	SIMNET protocol version used in the variant portion of the PDU
PDU TYPE	PDU type to follow in the variant portion of the packet
EXERCISE ID	Exercise generating PDU (important when multiple exercises on network)

Fire variant

- ATTACKER ID — Vehicle identification
  - Simulation address
    - Site
    - Host
  - Vehicle
- EVENT ID — For correlation with impact PDU
- BURST DESCRIPTOR
  - Projectile — Munition
  - Detonator — Detonator
  - Quantity — # of projectiles
  - Rate — Burst rate
- TARGET DESCRIPTOR
  - Target type
    - 0 Target unknown
    - 1 Target not a vehicle
    - 2 Target is a vehicle
  - Vehicle ID
- VELOCITY VECTOR — Velocity of the projectile
- LOCATION — World coordinates of origination of projectile
- PROJECTILE ID — Vehicle ID of projectile
  - Simulation address
    - Site
    - Host
  - Vehicle
- FIRE TYPE — Type of projectile
  - 1 Fire type shell
  - 2 Fire type missile
- If FIRE TYPE = shell
  - RANGE — Range of munition
  - SLEW RATE — rate
  - AMMO TYPE — Type of ammunition
- If FIRE TYPE = missile
  - TUBE — Tube from which missile was launched
- TIME STAMP — Time when PDU was issued

2.6 Impact PDU

An Impact is issued by a simulator when the flight of a projectile it is simulating ends. It ma or may not describe an impact between the projectile and a particular target vehicle.

FIELD SIZE (bits)	IMPACT PDU FIELDS		
8	PROTOCOL VERSION	8-bit unsigned integer	
8	PDU TYPE	8-bit unsigned integer	
8	EXERCISE ID	8-bit unsigned integer	
40	PADDING	40-bit unsigned integer	B
48	ATTACKER ID	Site - 16-bit unsigned integer	
		Host - 16-bit unsigned integer	
		Vehicle - 16-bit unsigned integer	
16	EVENT ID	16-bit unsigned integer	
96	BURST DESCRIPTOR	Projectile - 32-bit unsigned integer	
		Detonator - 32-bit unsigned integer	
		Quantity - 16-bit unsigned integer	
		Rate - 16-bit unsigned integer	
48	PROJECTILE ID	Site - 16-bit unsigned integer	
		Host - 16-bit unsigned integer	
		Vehicle - 16-bit unsigned integer	
8	FIRE RESULT	8-bit unsigned integer	B
8	PADDING	8-bit unsigned integer	
32	MOMENTUM	32-bit floating point	
32	ENERGY	32-bit floating point	

FIELD SIZE (bits)	IMPACT PDU CONTINUED	
32	DIRECTIONALITY	32-bit floating point
192	LOCATION (WORLD COORDINATES)	x - 64-bit floating point
		y - 64-bit floating point
		z - 64-bit floating point
64	RANGE	64-bit floating point
48	TARGET ID	Site - 16-bit unsigned integer
		Host - 16-bit unsigned integer
		Vehicle - 16-bit unsigned integer
16	VEHICLE COMPONENT	16-bit unsigned integer
96	IMPACT LOCATION (VEHICLE COORDINATES)	x - 32-bit floating point
		y - 32-bit floating point
		z - 32-bit floating point
96	TRAJECTORY (VEHICLE COORDINATES)	x - 32-bit floating point
		y - 32-bit floating point
		z - 32-bit floating point
32	TIME STAMP	32-bit unsigned integer
16	PK	16-bit integer

Total Impact PDU Size = 928 bits 8

Simulation PDU header information

PROTOCOL VERSION	SIMNET protocol version used in the variant portion of the PDU
PDU TYPE	PDU type to follow in the variant portion of the packet
EXERCISE ID	Exercise generating PDU (important when multiple exercises on network)

Impact variant

ATTACKER ID		Vehicle identification
	Simulation address	Site
		Host
	Vehicle	
EVENT ID		For correlation with fire PDU
BURST DESCRIPTOR		
	Projectile	Munition

Detonator	Detonator
Quantity	# of projectiles
Rate	Burst rate
PROJECTILE ID	Vehicle ID of projectile
Simulation address	Site
	Host
Vehicle	
FIRE RESULT	
14	Hit / Terminate / Kill
15	No target miss
16	Velocity gate miss
17	Gimbal limit miss
18	Ground impact miss
19	Low closure rate miss
20	Low velocity miss
21	Max time of flight miss
22	Safe-arm miss
23	Low probability of kill miss
24	Excessive miss distance
25	Target already killed
26	Line of sight miss (AIM-9)
27	Jettisoned
28	Terminated but not yet scored
MOMENTUM	Momentum of projectile
ENERGY	Energy of projectile at impact
DIRECTIONALITY	Directionality of projectiles explosion in steradians
LOCATION	Location of impact in world coordinates (meters)
RANGE	Range of projectile
TARGET ID	Vehicle ID of target
Simulation address	Site
	Host
Vehicle	
VEHICLE COMPONENT	Component struck by projectile
0	Vehicle component irrelevant
1	Hull component
2	Turret component
IMPACT LOCATION	Location of impact in vehicle coordinates
TRAJECTORY	Vehicle coordinates
TIME STAMP	Time when PDU was issued
PK	Probability of kill

2.7 Radar PDU

A Radar periodically issued by the simulator of a vehicle possessing a radar. The PDU's describe the location, and characteristics of the signals with the following data:

FIELD SIZE (bits)	RADAR PDU FIELDS		
8	PROTOCOL VERSION	8-bit unsigned integer	
8	PDU TYPE	8-bit unsigned integer	
8	EXERCISE ID	8-bit unsigned integer	
40	PADDING	40-bit unsigned integer	B
48	VEHICLE ID	Site - 16-bit unsigned integer	
		Host - 16-bit unsigned integer	
		Vehicle - 16-bit unsigned integer	E
8	# ILLUMED	8-bit unsigned integer	
8	PADDING	8-bit unsigned integer	
32	RADAR SYSTEM	32-bit integer	
8	RADAR MODE	8-bit unsigned integer	
8	PADDING	8-bit unsigned integer	
128	SWEEP	Azimuth Center - 32-bit floating point	
		Azimuth Width - 32-bit floating point	
		Elevation Center - 32-bit floating point	
		Elevation Width - 32-bit floating point	
32	POWER	32-bit integer	
32	TIME STAMP	32-bit unsigned integer	E

FIELD SIZE (bits)	RADAR PDU CONTINUED		
80 n	VEHICLE ID	Site - 16-bit unsigned integer	For Each Illumined Entity
		Host - 16-bit unsigned integer	
		Vehicle - 16-bit unsigned integer	
	RADAR DATA	32-bit integer	

Total Radar PDU Size = 368 + 80n bits | E

Simulation PDU header information

PROTOCOL VERSION — SIMNET protocol version used in the variant portion of the PDU
PDU TYPE — PDU type to follow in the variant portion of the packet
EXERCISE ID — Exercise generating PDU (important when multiple exercises on network)

Radar variant

VEHICLE ID — Vehicle identification
  Simulation address — Site
    Host
  Vehicle | E
ILLUMED — Number of vehicles illuminated by radar
RADAR SYSTEM — Bit field identifying radar system
  Radar System Category (Bits 28-31)
    0 Reserved (unused)
    1 Air-Based Fire Control
    2 Air-Based Search
    3 Ground-Based Fire Control
    4 Ground-Based Search
    5 Sea-Based Fire Control
    6 Sea-Based Search
  Radar System Subcategory(Bits 16-23 optional)
  Radar System ID (Bits 0-15)

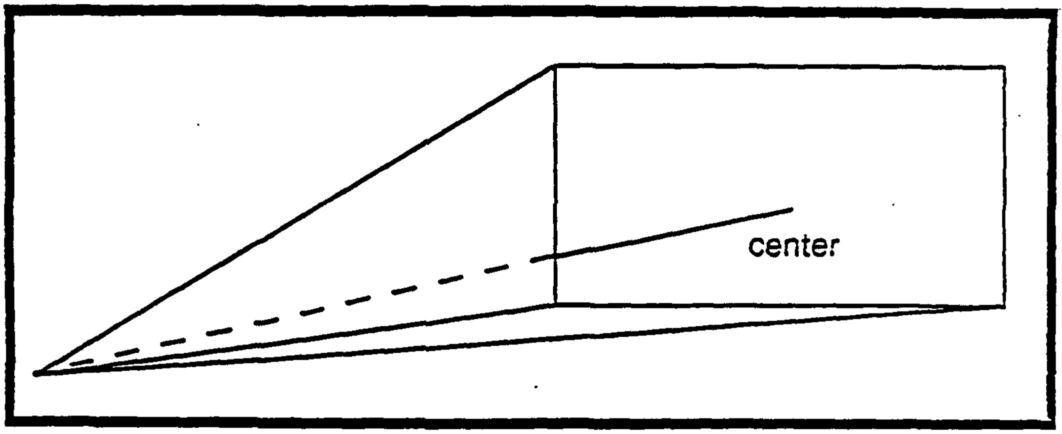
0	Reserved	14	HighLark
1	APG-66	15	AN/APS-125
2	APG-68	16	LN-66 HP
3	APG-63	17	AN/APS-166
4	APG-65	18	AN/APS-115
5	APG-70	19	AN/SPQ-9
6	JAYBIRB	20	AN/SPQ-9A
7	(Mig-31)	21	AN/SPG-60
8	(Mig-29)	22	AN/SPS-49
9	(Mig-27)	23	AN/SPS-55
10	(Su-27)	24	AN/SPS-67
11	AN/APY-2	25	AN/SPS-10

12	SUAWACS	26	SPY-1a
13	FoxFire		

RADAR MODE Current radar mode

1 Search
2 Doppler HPRF
3 Doppler MPRF
4 Doppler LPRF
5 Monopulse
6 Acquisition
7 Tracking
8 Track while scan
9 Terrain follow
10 Data link

AZIMUTH CENTER Azimuth center angle (degrees) | E
AZIMUTH WIDTH Azimuth width half angle (degrees) | E
ELEVATION CENTER Elevation center angle (degrees) | E
ELEVATION WIDTH Elevation width half angle (degrees) | E



RADAR CONE

RADAR POWER Average emitting power in decibel milliwatts
TIME STAMP Time when PDU was issued | E
RADAR TARGET LIST
Vehicle ID
Radar data
bits 24 - 31 -> Radar Mode pertaining to applicable Vehicle ID
bits 0 - 23 -> Specific Radar System/Radar Mode data (optional)
Might be : Polarization, Freq Hopping, Staggered PRF, etc]

NOTE: Due to Ethernet packet constraints, the limit on the number of radars per PDU is theoretically 100. However, in actual practice, the number is normally in the range of 1 to 10 radars. | E

2.8 Emitter PDU

An Em[illegible]er periodically issued by the simulator of a vehicle possessing an Emitter(s). The PDU's describe the location, and characteristics of the signals with the following data:

FIELD SIZE (bits)	EMITTER PDU FIELDS	
8	PROTOCOL VERSION	8-bit unsigned integer
8	PDU TYPE	8-bit unsigned integer
8	EXERCISE ID	8-bit unsigned integer
40	PADDING	40-bit unsigned integer
48	VEHICLE ID	Site - 16-bit unsigned integer
		Host - 16-bit unsigned integer
		Vehicle - 16-bit unsigned integer
16	# EMITTERS	16-bit integer
32	TIME STAMP	32-bit unsigned integer
256 n	EMITTER CLASS	16-bit unsigned integer
	DATABASE #	16-bit unsigned integer
	EMITTER MODE	16-bit unsigned integer
	EMITTER POWER	16-bit unsigned integer
	FREQUENCY	32-bit floating point
	CHANNEL	32-bit unsigned integer
	SWEEP	Azimuth Center - 32-bit floating point
		Azimuth Width - 32-bit floating point
		Elevation Center - 32-bit floating point
		Elevation Width - 32-bit floating point

For Each Emitter

Total Emitter PDU Size = 160 + 256n bits

Simulation PDU header information

PROTOCOL VERSION	SIMNET protocol version used in the variant portion of the PDU
PDU TYPE	PDU type to follow in the variant portion of the packet
EXERCISE ID	Exercise generating PDU (important when multiple exercises on network)

Emitter variant

VEHICLE ID		Vehicle identification	
	Simulation address	Site	
		Host	
	Vehicle		
# EMITTERS		Number of emitters on vehicle	| E
TIME STAMP		Time when PDU was issued	| E

For each emitter

EMITTER CLASS

0	Other	9	SHF
1	Sound	10	EHF
2	infrasonic2 |	11	Infrared
3	VHF	12	Visible
4	LF	13	Ultraviolet
5	MF	14	XRay
6	HF	15	Gamma Ray
7	VHF	16	Cosmic Ray
8	UHF		

DATABASE NUMBER

VHF	0x0001	ILS	0x0020	Jammer	0x1000
UHF	0x0002	AAI	0x0100		
TACAN	0x0010	IFF	0x0200		

EMITTER MODE

0	Transmit
1	Mode 1
2	Mode 2
3	Mode 3
4	Mode 4
5	Mode 4a
6	Mode 4b

EMITTER POWER	Average power of emission	
FREQUENCY	Frequency of emission	
CHANNEL	Emitter channel	
AZIMUTH CENTER	Azimuth center angle (degrees)	| E
AZIMUTH WIDTH	Azimuth width half angle (degrees)	| E
ELEVATION CENTER	Elevation center angle (degrees)	| E
ELEVATION WIDTH	Elevation width half angle (degrees)	| E

NOTE: Due to Ethernet packet constraints, the limit on the number of emitters per PDU is theoretically 40. However, in actual practice, the number is normally in the range of 1 to 10 emitters. | E

2.9 Freeze PDU

The freeze PDU is used to both freeze and unfreeze. It can be used both globally and individually.

FIELD SIZE (bits)	FREEZE PDU FIELDS		
8	PROTOCOL VERSION	8-bit unsigned integer	
8	PDU TYPE	8-bit unsigned integer	
8	EXERCISE ID	8-bit unsigned integer	
40	PADDING	40-bit unsigned integer	
8	FREEZE MODE	8-bit unsigned integer	
8	PADDING	8-bit unsigned integer	
32	TIME STAMP	32-bit unsigned integer	
16	# VEHICLES	16-bit unsigned integer	
48 n	VEHICLE ID	Site - 16-bit unsigned integer Host - 16-bit unsigned integer Vehicle - 16-bit unsigned integer	For each Selected Vehicle

E

Total Freeze PDU Size = 128 + 48n bits

B

Simulation PDU header information

PROTOCOL VERSION	SIMNET protocol version used in the variant portion of the PDU
PDU TYPE	PDU type to follow in the variant portion of the packet
EXERCISE ID	Exercise generating PDU (important when multiple exercises on network)

Freeze variant

FREEZE MODE

- 0 Unfreeze
- 1 Freeze

TIME STAMP	Time PDU was issued
# VEHICLE	Number of vehicles to change freeze state (Note: use 0 for global)

VEHICLE ID ARRAY		Optional array of vehicle ID's if selectively changing freeze state
	Simulation address	Site Host Vehicle

AL0692-009 Rev.E 1 December 1992

APPENDIX A

SIMNET 6.6.1+ NETWORK PROTOCOLS FOR THE TRUE & WAR BREAKER PROGRAMS

Guise Definitions

*** Chaff ***

Chaff:	0x4100400	

*** Flares ***

MJU-7:	0x8100407	
MJU-10:	0x810040a	

*** AAA ***

ZSU23_4M:	0x28842821	

*** US Air Vehicles ***

A-10:	0x24820802	
F-16A:	0x24821021	
F-16B	0x24821041	
F-16C:	0x24821061	
F-16D:	0x24821081	
F-14A:	0x24821821	|C
F-14D:	0x24821841	|C
F-15E:	0x24822021	|C
F-15C:	0x24822042	|C
F-20:	0x24822801	|C
F-4S:	0x24823021	|C
F-5F:	0x24823821	|C
A-7:	0x24824001	|C
Pioneer RPV:	0x24824003	|C
E-8A:	0x24824803	|C
TR-1A:	0x24825003	|C
TR-1B:	0x24825023	|C
KC-10A:	0x24825807	|C
E-3:	0x24826008	|C
EF-111A:	0x24826809	|C
F-4G:	0x24827009	|C
F-18:	0x24827801	|C
A-6:	0x24828001	|C
B-52G:	0x24828804	|C
F-22:	0x24828801	|C
AH-64:	0x25820802	|C
OH-58D:	0x25821003	|C
OH-58C:	0x25821023	|C
AH-1:	0x25821802	|C
UH-60:	0x25822005	|C
CH-47:	0x25822806	|C

*** US Ground Vehicles ***

M1:	0x2882080c	|C
M60:	0x2882100c	|C
M2:	0x28821022	|C

APPENDIX A
(Continued)

Guise Definitions

M3:	0x28821047	|C
M113A2:	0x28821822	|C
M113 FIST:	0x28821842	|C
M113 ambulance:	0x28821862	|C
M113 engineer:	0x28821882	|C
M577:	0x28821843	|C
M106A1:	0x28821865	|C
M109:	0x28822004	|C
M88A1:	0x28822828	|C
ADATS:	0x28823001	|C
LOSAT:	0x2882380a	|C
M35A2:	0x2a020829	|C
M977:	0x2a021029	|C
M978:	0x2a021049	|G
M57:	0x2a82080d	|C
M128:	0x2a82100d	|C
M57:	0x2a82080d	|C
M128:	0x2a82100d	|C
M58A1:	0x2a82180d	|C
PALLET:	0x2a822009	|C
A22:	0x26820809	|C
HUMMV:	0x2a02180f	|C
M270:	0x28820806	|C
M901 (Patriot launcher):	0x2a820001	|C
MPQ53 (Patriot radar):	0x2a820010	|C

*** US Sea Vehicles ***

LHD1:	0x30820822	|C
LHD2:	0x30820842	|C
LCC19:	0x30821024	|C
LHA1:	0x30821822	|C
LPH2:	0x30822022	|C
LKA113:	0x30822823	|C
DD963:	0x3302082d	|C
DD976:	0x3302084d	|C
DD964:	0x3302086d	|C
CG47:	0x3302102b	|C
CG42:	0x3302104b	|C
CG52:	0x3302104b	|C
CVN68:	0x33021829	|C
CVN69:	0x33021849	|C
CVN70:	0x33021869	|C
BB71:	0x33022028	|C
BB72:	0x33022048	|C
BB73:	0x33022068	|C
BB74:	0x33022088	|C
PHM1:	0x3302b032	|C
PHM2:	0x3302b052	|C
CG26:	0x3302282b	|C
CG16:	0x3302302b	|C
CG17:	0x3302304b	|C
CGN38:	0x3302382b	|C
CGN36:	0x3302402b	|C
CGN37:	0x3302404b	|C
CGN35:	0x3302482b	|C
CGN25:	0x3302502b	|C
CGN9:	0x3302582b	|C
CV63:	0x33026029	|C
CV64:	0x33026049	|C
CV65:	0x33026069	|C
CV59:	0x33026829	|C
CV41:	0x33027029	|C
CV42:	0x33027049	|C
CVN65:	0x33027829	|C
DDG51:	0x3302802d	|C
DDG52:	0x3302804d	|C
DDG993:	0x3302882d	|C
DDG994:	0x3302884d	|C
DDG40:	0x3302902d	|C
DDG2:	0x3302982d	|C
FF1052:	0x3302a031	|C
FF1040:	0x3302a831	|C
FFG7:	0x3302b831	|C

APPENDIX A
(Continued)

Guise Definitions

FFG1:	0x3302c031	|C
SSBN726:	0x32820827	|C
SSBN727:	0x32820847	|C
SSBN728:	0x32820867	|C
SSBN640:	0x32821027	|C
SSN637:	0x32821826	|C
SSN585:	0x32822026	|C
SSN590:	0x32822046	|C
SSN688:	0x32822826	|C
SSN689:	0x32822846	|C
SSN753:	0x32822866	|C
SS580:	0x32823026	|C
AD41:	0x3182082c	|C
AD42:	0x3182084c	|C
AFS1:	0x3182102a	|C

*** USSR Air Vehicles ***

SU-25:	0x24840802	
SU-27:	0x24842002	
TU-20:	0x24841004	|C
TU-22:	0x24841804	|C
Mig-23:	0x24841001	
Mig-27:	0x24841801	
Mig-21:	0x24842001	|C
Mig-25:	0x24842801	
Mig-29:	0x24843001	|C
Mig-31:	0x24843801	|C
F-1:	0x24844001	|C
IL-76:	0x2484080a	|C
Mi-24:	0x25840802	|C
Mi-24D:	0x25840802	|C
Mi-24F:	0x25840822	|C
Mi-28:	0x25841002	|C
Mi-8:	0x25841802	|C
Mi-17:	0x25842002	|C
SA-342:	0x25842802	|C
MBB-150:	0x25843002	|C
Mi-6:	0x25843805	|C
Mi-26:	0x25844006	|C

*** USSR Ground Vehicles ***

BRM:	0x28844007	|C
BRDM2:	0x29044807	|C
ACRV:	0x28843003	|C
GAZ66:	0x2a040809	|C
URAL375C:	0x2a041029	|C
URAL375F:	0x2a041049	|C
ZIL157:	0x2a041809	|C
SU refueler:	0x2a042009	|C
water carrier:	0x2a042809	|C
UAZ469:	0x2a04080f	|C
PMR3:	0x2a84000d	|C
MICLIC:	0x2a84080d	|C
GMZ:	0x2a04000d	|C
PALLET:	0x2a840809	|C
A22:	0x26840809	|C
BREM1:	0x28840848	|C
T80:	0x2884380c	|C
T72M:	0x2884082c	|C
T64:	0x2884480c	|C
T62:	0x2884500c	|C
T55:	0x2884580c	|C
T54:	0x2884600c	|C
Chieftain:	0x2884680c	|C
BRDM:	0x29040802	|C
BMP1:	0x28841022	|C
BMP1K:	0x28841043	|C
BTR80:	0x2a041802	|C
BMP2:	0x28842002	|C
MTLB:	0x28842822	|C
MTLB ambulance:	0x28842842	|C
BM21:	0x2a040826	|C
M1943:	0x2a840805	|C

APPENDIX A
(Continued)

Guise Definitions

105 HOWITZER:	0x2a840804	|C
D20:	0x2a841004	|C
2S1:	0x28841804	|C
SCUD-B:	0x2904082e	|C
FROG-7:	0x2904102e	|C
MAZ543 (SCUD TEL):	0x2a04182e	|C
ZSU23_4M:	0x28842821	|C
ZSU57_2:	0x28840821	|C
SA-02:	0x2a041021	|C
SA-03:	0x2a041821	|C
SA-04:	0x29042021	|C
SA-06:	0x29043021	|C
SA-07:	0x29043821	|C
SA-08:	0x29044021	|C
SA-09:	0x29044821	|C
SA-13:	0x29046821	|C
ROLAND:	0x29047021	|C
I_HAWK:	0x29047821	|C
S_60:	0x2a840821	|C

*** USSR Water Vehicles ***

ALFA:	0x32840806	|C

*** German Vehicles ***

LEO2:	0x2886080c	|C
MARDER:	0x28861002	|C

*** Other Structures ***

BUILDING1:	0x60000001	|C
Minefield Marker:	0x60000002	|C
Breached Lane Marker:	0x60000003	|C
Broken Stone Bridge:	0x60000004	|C
Broken Steel Bridge:	0x60000005	|C
Broken Low Bridge:	0x60000006	|C
Long Track Radar:	0x60000007	|C
Bar Lock Radar:	0x60000008	|C
Arrow:	0x60000009	|C

*** Colored Cubes ***

Black Cube:	0x6000000a	|C
White Cube:	0x6000000b	|C
Red Cube:	0x6000000c	|C
Orange Cube:	0x6000000d	|C
Yellow Cube:	0x6000000e	|C
Green Cube:	0x6000000f	|C
Blue Cube:	0x60000010	|C
Violet Cube:	0x60000011	|C

*** Logistics Structures ***

Ammo Crate:	0x60000012	|C
Tent:	0x60000013	|C

*** 73 Easting Battle Reenactment Buildings ***

Barracks A:	0x60000015	|C
Barracks B:	0x60000016	|C
Barracks C:	0x60000017	|C

APPENDIX A
(Continued)

Guise Definitions

Barracks D:	0x60000018	|C
Barracks E:	0x60000019	|C
Barracks F:	0x6000001a	|C
Barracks G:	0x6000001b	|C
Barracks H:	0x6000001c	|C
Barracks I:	0x6000001d	|C
Barracks J:	0x6000001e	|C
Barracks K:	0x6000001f	|C
Log Site:	0x60000020	|C

*** Buildings ***

Cross T:	0x60000021	|C
Large Residential Area:	0x60000022	|C
Large Commercial Area:	0x60000023	|C
Hex A:	0x60000024	|C
Brick A:	0x60000025	|C

*** Non-Buildings ***

Smoke Stack B:	0x60000026	|C
Minaret A:	0x60000027	|C
Corral_C:	0x60000028	|C
Oil Tank A:	0x60000029	|C

*** Warehouses ***

Warehouse A:	0x6000002a	|C
Warehouse D:	0x6000002b	|C
Warehouse G:	0x6000002c	|C
Warehouse W:	0x6000002d	|C
Warehouse Y:	0x6000002e	|C

*** Factories ***

Factory A:	0x6000002f	|C

*** Garages ***

Garage Open W:	0x6000[illegible]030	|C
Garage Open Y:	0x6000[illegible]031	|C
Power Transformer Station:	0x6000[illegible]032	|C

*** Houses ***

House RI:	0x60000033	|C
House S:	0x60000034	|C
House T:	0x60000035	|C

*** Railroad Structures ***

Railroad car A:	0x60000036	|C
Railroad car B:	0x60000037	|C
Railroad garage A:	0x60000038	|C

*** Water Processing Structures ***

Water Pump Station A:	0x60000039	|C

APPENDIX A
(Continued)

Guise Definitions

Water Pump Station B:	0x6000003a	|C
Water Plant:	0x6000003b	|C
Water Tower S:	0x6000003c	|C

*** Highway Structures ***

Highway Deck 100m:	0x6000003d	|C

*** Life Forms ***

Friendly DI:	0x80000001	|C
Enemy DI:	0x80000002	|C
Friendly Group DI:	0x80000003	|C
Enemy Group DI:	0x80000004	|C
Friendly Line DI:	0x80000005	|C
Enemy Line DI:	0x80000006	|C
Friendly Column DI:	0x80000007	|C
Enemy Column DI:	0x80000008	|C

*** Diffuse Matter ***

Medium Atmospheric Cloud:	0xd0904100	|C
Medium Smoke Cloud:	0xd1204100	|C
Medium Flame:	0xd1904100	|C

*** Fuel ***

Fuel:	0x46000000	|C

*** US ammuniton ***

M904:	0x42010420	|C
M557:	0x42b10420	|C
M513:	0x42b20420	|C
M739:	0x43010420	|C
M728:	0x43030420	|C
M603:	0x42120420	|C
M33:	0x48280420	|C
M50:	0x48240420	|C
M791:	0x48380420	|C
M792:	0x48340420	|C
M789:	0x48340440	|C
M392A2:	0x48b80421	|C
M456A1:	0x48bb0421	|C
M329:	0x48b40420	|C
M107:	0x49040420	|C
M855:	0x48180421	|C
M856:	0x48180422	|C
Mk82:	0x4c510420	
GBU-10:	0x4d490400	|C
GBU-12:	0x4c590420	|C
GBU-16:	0x4ca90440	|C
GBU-15-1B:	0x4db90460	|C
GBU-15-2B:	0x4db90461	|C

*** US Missiles ***

TOW:	0x442b0420	|C
M47:	0x442b0440	|C
Hellfire:	0x442b0460	|C
Maverick:	0x442b0480	|C
Sidewinder:	0x44140420	

APPENDIX A
(Continued)

Guise Definitions

ADATS:	0x44140440	|C
Stinger:	0x44140460	|C
Tomahawk:	0x448b0420	
Patriot:	0x44340440	|C
AIM-9L:	0x44140421	
AIM-9M:	0x44140422	
AIM-9P:	0x44140423	
AIM-9J:	0x44140424	
AIM-9D:	0x44140425	
AIM-9G:	0x44140426	
AIM-9H:	0x44140427	
Hawk:	0x44140441	|C
AIM-7M:	0x44140480	
AIM-7L:	0x44140481	
AIM-7F:	0x44140482	
AIM-7E:	0x44140483	
AIM-120A:	0x44140484	|C

*** US Mines ***

M15:	0x4e110421	|C
M19:	0x4e110441	|C
M21:	0x4e110461	|C
M741:	0x4e110481	|C
M718:	0x4e1104a1	|C
M75:	0x4e1104c1	|C
M14:	0x4e210421	|C
M18A1:	0x4e210441	|C
M16A2:	0x4e210461	|C
M731:	0x4e210481	|C
M692:	0x4e2104a1	|C
M74:	0x4e2104c1	|C
M56:	0x4e1104c1	|C

*** US Rockets ***

Hydra70 M151:	0x44240420	|C
Hydra70 M261:	0x44220420	|C
Hydra70 M255:	0x44120420	|C
M73:	0x484b0420	|C
Flechette 60:	0x48180440	|C
M433:	0x42610420	|C
M439:	0x42630420	|C
M26:	0x44020440	|C
M26 bomblets:	0x481b0420	|C
GPBomb:	0x4cb10420	|C

*** USSR ammuniton ***

UV-32-57:	0x44240820	|C
SNEB-68:	0x44240840	|C
UV-20-80:	0x44240860	|C
S-5:	0x48640820	|C
STYX-C:	0x44640820	|C
Songster:	0x442b0820	|C
Spandrel:	0x442b0840	|C
Spiral:	0x442b0860	|C
SwatterC:	0x442b0880	|C
HOT:	0x442b08a0	|C
Mistral:	0x44140800	|C
Gremlin:	0x44140820	|C
ALHUSAYN (SCUD):	0x44840820	|C
125HEAT:	0x48db0820	|C
125SABOT:	0x48d80820	|C
73HEAT:	0x488b0820	|C
30HE:	0x48340820	|C
20AP:	0x48280880	|C
23AP:	0x482808a0	|C
30SABOT:	0x48380820	|C
145MG:	0x48280820	|C
127AA:	0x48280840	|C
127MG:	0x48280860	|C
FAB250:	0x4c510820	|C
FAB500:	0x4cb10820	|C

APPENDIX A
(Continued)

Guise Definitions

Tank Internal Explosion:	0x4cb10840	|C
PC Internal Explosion:	0x4cb10841	|C
Missile Carrier Internal Explosion:	0x4cb10842	|C
Fuel Truck I nternal Explosion:	0x4cb10843	|C
Ammo Crate Internal Explosion:	0x4cb10845	|C
Vehicle Shrinking Smoke:	0x481b0820	|C
Fuel Site Shrinking Smoke:	0x481b0821	|C
Ammo Site Shrinking Smoke:	0x481b0822	|C
SA-2 missile:	0x441408a2	|D
SA-3 missile:	0x441408a3	|D
SA-4 missile:	0x441408a4	|C
SA-6 missile:	0x441408a6	|C
SA-7 missile:	0x441408a7	|C
SA-8 missile:	0x441408a8	|C
23mm:	0x44180901	|C
30mm:	0x44180902	|C
57mm:	0x44180903	|C

*** German Munitions ***

120HEAT:	0x48cb0c20	|C
120SABOT:	0x48c80c20	|C
20SABOT:	0x48280c20	|C
Milan:	0x442b0c20	|C

*** USSR Mines ***

TMD-B:	0x4e110820	|C
YAM-5:	0x4e110840	|C
TM46:	0x4e110861	|C
TMN46:	0x4e110881	|C
TM57:	0x4e1108a0	|C
TM62:	0x4e1108c0	|C
TMK2:	0x4e1108e0	|C
POMZ2:	0x4e210820	|C
OMZ3:	0x4e210840	|C
MON50:	0x4e210861	|C
MON100:	0x4e210862	|C
MON200:	0x4e210863	|C
PMN:	0x4e210880	|C
PMK40:	0x4e2108a0	|C

APPENDIX B

SIMNET 6.6.1+ NETWORK PROTOCOLS FOR THE TRUE & WAR BREAKER PROGRAMS

PDU TYPE NUMBERS

PDU	TYPE NUMBER
ACTIVATE REQUEST	1
ACTIVATE RESPONSE	2
DEACTIVATE REQUEST	3
VEHICLE APPEARANCE	5
FIRE	7
IMPACT	8
RADAR	30
EMITTER	31
FREEZE	33

APPENDIX C6: PERSISTENT OBJECT PROTOCOL

Naval Research Laboratory, Contract Number: N00014-92-C-2150
Data Item Number: A001, ModSAF B Software Documentation

LibPO

Persistent Object Library

Joshua E. Smith
Anthony J. Courtemanche

$Revision: 1.33 $

Table of Contents

1 Overview

The libpo library provides a simple interface to a shared database of "Persistent Objects." See Chapter 9 [Protocol Specification], page 49 for more on the characteristics of persistent objects and protocol specifics.

In many cases, two versions of handlers or functions exist: one refering to "object" and the other refering to "entry". In this context an entry is one world state (see Section 9.1.4 [World State], page 49) of an object. Using `po_set_world_state`, the application can specify the world state of objects it is interested in (the default is the Real Time world state). Thereafter, the application can refer only to objects and libpo will manage the creation and modification of entries.

The program xtest.c is an xwindows based test program which creates multiple databases and allows the user to experiment with the various functions of libpo. See Chapter 8 [Using Xtest], page 47.

2 Usage

The software library 'libpo.a' should be built and installed in the directory 'common/lib/'. You will also need the header file 'libpo.h' which should be installed in the directory 'common/include/libinc/'. If these files are not installed, you need to do a 'make' in the libpo source directory. If these files are already built, you can skip the section on building libpo.

2.1 Building Libpo

The libpo source files are found in the directory 'common/libsrc/libpo'. 'RCS' format versions of the files can be found in '/nfs/common_src/libsrc/libpo'.

If the directory 'common/libsrc/libpo' does not exist on your machine, you can create it using the following commands (assuming the 'common' software structure is present):

```
# cd common/libsrc
# mkdir libpo            ; you may need root privilege to do this
# cd libpo
# ln -s common/tools/make.config make.config
# ln -s common/tools/make.librules make.librules
# touch make.depend
# ln -s /nfs/common_src/libsrc/libpo RCS
```

To build and install the library, do the following:

```
# cd common/libsrc/libpo
# co RCS/*,v
# make
```

This should compile the library 'libpo.a' and install it and the header file 'libpo.h' in the standard directories. If any errors occur during compilation, you may need to adjust the source code or 'Makefile' for the platform on which you are compiling. libpo should compile without errors on the following platforms:

- Mips
- SGI Indigo
- Sun Sparc

2.2 Linking with Libpo

Libpo can be linked into the an application program with the following link time flags: '`ld [source .o files] -Lcommon/lib -lpo -ltime -lrandom -lcallback`'. If your compiler does not support '`-L`' syntax, you can use the archive explicitly:
'`ld [source .o files] common/lib/libpo.a`'.

Libpo depends on the following common libraries:

- librandom
- libtime
- libcallback

3 Data Types

The two primary data types an application will need to use with libpo are `PO_DATABASE` and `PO_DB_ENTRY`.

An application will typically create one `PO_DATABASE` at startup time with `po_create()` and will remove it at exit time with `po_destroy()`. This database is passed to all libpo routines.

An application will deal with many `PO_DB_ENTRY` structures. A single `PO_DB_ENTRY` describes the state of an object in the shared database at one time. An application creates new objects in the database using `po_create_object()` or `po_create_world_state()`. Applications learn about new objects created by other users of the shared database through the events described below.

4 Events and Event Handlers

When an application first opens the database with **po_create()**, it provides a send handler which is invoked by libpo in order to send PDU's to the network. Also, several events are created. These events can be used to attach user handlers for certain events.

The sections below describe the event handler functions by including a synopsis and a description.

4.1 send_handler

```
void send_handler(pdu, size, user_data)
    PersistentObjectPDU *pdu;
    uint32              size;
    PO_USER_DATA_TYPE   user_data;
```

send_handler is called by libpo to send a packet.

If the application is using the libassoc to send packets, the send handler might be coded:

```
void send_handler(pdu, size, assoc_handle)
    PersistentObjectPDU *pdu;
    uint32              size;
    int                 assoc_handle;
{
    extern int g_exercise_id;
    extern assoc_error_handler();

    ret = AssocSendDatagram(assoc_handle, pdu, size, g_exercise_id,
                            persistentObjectProtocolNumber);
    if (ret<0)
      assoc_error_handler(ret);
}
```

4.2 query_handler

```
void query_handler(entry, user_data)
    PO_DB_ENTRY       *entry;
    PO_USER_DATA_TYPE user_data;
```

A query_handler is called in response to po_query_for_current_objects and in response to po_query_for_all_entries. Each entry will contain a describeObjectPDU. Note that it *is not safe* to delete objects or entries in a query_handler routine. Instead, an application should compile a list of objects or entries to delete and delete them all with po_delete_objects or po_delete_entries.

4.3 overlay_confirmation_handler

```
int32 overlay_confirmation_handler(name, user_data)
    char              *name;
    PO_USER_DATA_TYPE user_data;
```

The overlay_confirmation_handler is called during the reading of files via po_load_file. It is called once for each new overlay encountered in the file. A return value of TRUE confirms that the overlay can be loaded. A return value of FALSE indicates that file loading should be aborted. This can be used to prevent the loading of overlays which are already loaded into the machine. See Sec : 6.34 [po'load'file], page 36.

4.4 new_simulator_event_handler

The db->new_simulator_event is a libcallback event which can be accessed from an open po database db. Applications may attach a new_simulator_event_handler to this event via callback_register_handler (see section "callback_register_handler" in *LibCallback Programmer's Manual*).

```
void new_simulator_event_handler(entry, user_data)
    PO_DB_ENTRY *entry;
    ADDRESS      user_data;
```

new_simulator_event_handler is called when a new active (see Section 9.1.2 [Active Simulator], page 49) user of the shared database appears on the network. The entry will contain a simulatorPresentPDU. The application can use this handler to keep the user informed of what other machines might be modifying the database.

Note that the new_simulator_event is destroyed after a po database is destroyed.

4.5 simulator_gone_event_handler

The `db->simulator_gone_event` is a libcallback event which can be accessed from an open po database `db`. Applications may attach a `simulator_gone_event_handler` to this event via `callback_register_handler` (see section "callback_register_handler" in *LibCallback Programmer's Manual*).

```
void simulator_gone_event_handler(entry, user_data)
    PO_DB_ENTRY *entry;
    ADDRESS      user_data;
```

`simulator_gone_event_handler` is called when an active (see Section 9.1.2 [Active Simulator], page 49) user of the shared database disappears from the network. The `entry` will contain the last `simulatorPresentPDU` sent by that simulator. The application can use this handler to keep the user informed of what other machines might be modifying the database.

Note that the `simulator_gone_event` is destroyed after a po database is destroyed.

4.6 new_entry_event_handler

The `db->new_entry_event` is a libcallback event which can be accessed from an open po database `db`. Applications may attach a `new_entry_event_handler` to this event via `callback_register_handler` (see section "callback_register_handler" in *LibCallback Programmer's Manual*).

```
void new_entry_event_handler(entry, user_data)
    PO_DB_ENTRY *entry;
    ADDRESS      user_data;
```

`new_entry_event_handler` is called when a new entry is added to the shared database. The `entry` will contain a `describeObjectPDU`. A new entry can describe a new object, or it can give new world state information (see Section 9.1.4 [World State], page 49) to an existing object. This is a low-level handler; most applications can rely instead on the `new_object_event_handler`.

Note that the `new_entry_event` is destroyed after a po database is destroyed.

4.7 entry_changed_event_handler

The **db->entry_changed_event** is a libcallback event which can be accessed from an open po database **db**. Applications may attach a **entry_changed_event_handler** to this event via **callback_register_handler** (see section "callback_register_handler" in *LibCallback Programmer's Manual*).

```
void entry_changed_event_handler(entry, user_data)
    PO_DB_ENTRY *entry;
    ADDRESS      user_data;
```

entry_changed_event_handler is called when an entry changes its data part. The **entry** will contain a **describeObjectPDU**. This is a low-level handler; most applications can rely instead on the **object_changed_event_handler**.

Note that the **entry_changed_event** is destroyed after a po database is destroyed.

4.8 entry_gone_event_handler

The **db->entry_gone_event** is a libcallback event which can be accessed from an open po database **db**. Applications may attach a **entry_gone_event_handler** to this event via **callback_register_handler** (see section "callback_register_handler" in *LibCallback Programmer's Manual*).

```
void entry_gone_event_handler(entry, user_data)
    PO_DB_ENTRY *entry;
    ADDRESS      user_data;
```

entry_gone_event_handler is called when an entry is removed from the shared database. The **entry** will contain the last **describeObjectPDU** sent regarding the **entry**. This is a low-level handler; most applications can rely instead on the **object_gone_event_handler**.

Note that the **entry_gone_event** is destroyed after a po database is destroyed.

4.9 new_object_event_handler

The **db->new_object_event** is a libcallback event which can be accessed from an open

po database db. Applications may attach a **new_object_event_handler** to this event via **callback_register_handler** (see section "callback_register_handler" in *LibCallback Programmer's Manual*).

```
void new_object_event_handler(entry, user_data)
    PO_DB_ENTRY *entry;
    ADDRESS     user_data;
```

new_object_event_handler is called when a new object appears in the current world state (see Section 9.1.4 [World State], page 49). The **entry** will contain a describeObjectPDU.

Note that the **new_object_event** is destroyed after a po database is destroyed.

4.10 object_changed_event_handler

The **db->object_changed_event** is a libcallback event which can be accessed from an open po database db. Applications may attach a **object_changed_event_handler** to this event via **callback_register_handler** (see section "callback_register_handler" in *LibCallback Programmer's Manual*).

```
void object_changed_event_handler(entry, user_data)
    PO_DB_ENTRY *entry;
    ADDRESS     user_data;
```

object_changed_event_handler is called when an object changes with respect to the current world state (see Section 9.1.4 [World State], page 49). The entry will contain a describeObjectPDU.

Note that the **object_changed_event** is destroyed after a po database is destroyed.

4.11 object_gone_event_handler

The **db->object_gone_event** is a libcallback event which can be accessed from an open po database db. Applications may attach a **object_gone_event_handler** to this event via **callback_register_handler** (see section "callback_register_handler" in *LibCallback Programmer's Manual*).

```
void object_gone_event_handler(entry, user_data)
    PO_DB_ENTRY *entry;
    ADDRESS      user_data;
```

`object_gone_event_handler` is called when an object disappears with respect to the current world state (see Section 9.1.4 [World State], page 49). The entry will contain the previous `describeObjectPDU` regarding the object.

Note that the `object_gone_event` is destroyed after a po database is destroyed.

4.12 object_change_failed_event_handler

The `db->object_change_failed_event` is a libcallback event which can be accessed from an open po database `db`. Applications may attach a `object_change_failed_event_handler` to this event via `callback_register_handler` (see section "callback_register_handler" in *LibCallback Programmer's Manual*).

```
void object_change_failed_event_handler(entry, user_data)
    PO_DB_ENTRY *entry;
    ADDRESS      user_data;
```

`object_change_failed_event_handler` can be called when two simulators change the same object at roughly the same time. The entry will contain the `describeObjectPDU` with the new information, which can be passed to `po_entry_owner()` to find the identity of the other simulator.

Note that the `object_change_failed_event` is destroyed after a po database is destroyed.

4.13 delete_all_event_handler

The `db->delete_all_event` is a libcallback event which can be accessed from an open po database `db`. Applications may attach a `delete_all_event_handler` to this event via `callback_register_handler` (see section "callback_register_handler" in *LibCallback Programmer's Manual*).

```
void delete_all_event_handler(entry, user_data)
    PO_DB_ENTRY *entry;
    ADDRESS      user_data;
```

delete_all_event_handler is called when some simulator on the network explicitly deletes all objects in the shared database. There is nothing an application can do to stop the deletion once it has begun. This handler is called before the delete all occurs, so backing up objects to disk can be done in the context of this handler. The **entry** will contain a **simulatorPresentPDU** which can be passed to po_entry_owner() to find the identity of the simulator which caused the deletion.

Note that the delete_all_event is destroyed after a po database is destroyed.

4.14 world_state_changing_event_handler

The db->world_state_changing_event is a libcallback event which can be accessed from an open po database db. Applications may attach a world_state_changing_event_handler to this event via callback_register_handler (see section "callback_register_handler" in *LibCallback Programmer's Manual*).

```
void world_state_changing_event_handler(entry, user_data)
    PO_DB_ENTRY *entry;
    ADDRESS      user_data;
```

world_state_changing_event_handler is called before the current world state (see Section 9.1.4 [World State], page 49) of the database is changed. This can be caused by an explicit request from the application to change the current world state, a request from the network for applications to set the world state, or by the current world state being deleted from the network. The entry will contain a describeObjectPDU describing the new current world state.

Note that the world_state_changing_event is destroyed after a po database is destroyed.

4.15 world_state_changed_event_handler

The db->world_state_changed_event is a libcallback event which can be accessed from an open po database db. Applications may attach a world_state_changed_event_handler to this event via callback_register_handler (see section "callback_register_handler" in *LibCallback Programmer's Manual*).

```
void world_state_changed_event_handler(entry, user_data)
    PO_DB_ENTRY *entry;
    ADDRESS      user_data;
```

wor_:_state_changed_event_handler is called when the current world state (see Sectior .1.4 [World State], page 49) of the database is changed. This can be caused by an explicit request :rom the application to change the current world state, a request from the network for applications to set the world state, or by the current world state being deleted from the network. The entry will contain a describeObjectPDU describing the new current world state.

Note that the world_state_changed_event is destroyed after a po database is destroyed.

4.16 animation_event_handler

The db->animation_event is a libcallback event which can be accessed from an open po database db. Applications may attach a animation_event_handler to this event via callback_register_handler (see section "callback_register_handler" in *LibCallback Programmer's Manual*).

```
void animation_event_handler(pdu, user_data)
    PersistentObjectPDU *pdu;
    ADDRESS             user_data;
```

animation_event_handler is called when another simulator on the network has issued a SetWorldStatePDU.

Note that the animation_event is destroyed after a po database is destroyed.

4.17 animation_timeout_event_handler

The db->animation_timeout_event is a libcallback event which can be accessed from an open po database db. Applications may attach a animation_timeout_event_handler to this event via callback_register_handler (see section "callback_register_handler" in *LibCallback Programmer's Manual*).

```
void animation_timeout_event_handler(user_data)
    ADDRESS user_data;
```

animation_timeout_event_handler is called when a stream of SetWorldStatePDU is interrupted (probably because of a missed packet).

Note that the `animation_timeout_event` is destroyed after a po database is destroyed.

4.18 project_event_handler

The `db->project_event` is a libcallback event which can be accessed from an open po database db. Applications may attach a `project_event_handler` to this event via `callback_register_handler` (see section "callback_register_handler" in *LibCallback Programmer's Manual*).

```
void project_event_handler(vehicleID, do_project, user_data)
    ObjectID *vehicleID;
    int32    do_project;
    ADDRESS  user_data;
```

`project_event_handler` is called when the `stealthControllerClass` object regarding the stealth with the passed `vehicleID` first identifies the receiving simulator as the controller (`do_project == TRUE`), or no longer identifies the receiving simulator as the controller (`do_project == FALSE`).

Note that the `project_event` is destroyed after a po database is destroyed.

4.19 exercise_initialization_event_handler

The `db->exercise_initialization_event` is a libcallback event which can be accessed from an open po database db. Applications may attach a `exercise_initialization_event_handler` to this event via `callback_register_handler` (see section "callback_register_handler" in *LibCallback Programmer's Manual*).

```
void exercise_initialization_event_handler(exercise_data, user_data)
    ExerciseInitializerClass *exercise_data;
    ADDRESS                  user_data;
```

`exercise_initialization_event_handler` is called when the `ExerciseInitializerClass` object is first received or changed. The data in the object is passed to the handler as `exercise_data`.

Note that the `exercise_initialization_event` is destroyed after a po database is destroyed.

4.20 packets_missed_event_handler

The db->packets_missed_event is a libcallback event which can be accessed from an open po database db. Applications may attach a packets_missed_event_handler to this event via callback_register_handler (see section "callback_register_handler" in *LibCallback Programmer's Manual*).

```
void packets_missed_event_handler(entry, num_missed, user_data)
    PO_DB_ENTRY *entry;
    uint32       num_missed;
    ADDRESS      user_data;
```

packets_missed_event_handler is called when libpo detects that it has missed packets from a simulator. The entry identifies the simulator which sent the packets. The num_missed field identifies the minimum number of packets missed. Although it is possible that a simulator will not detect missing packets, it is unlikely. This handler is provided only to aid in debugging. It is not expected that an application would do anything more than print a message or keep a counter when packets are missed. po_entry_owner can be passed with entry to get the symbolic name of the simulator.

Note that the packets_missed_event is destroyed after a po database is destroyed.

5 Global Variables

The sections below describe the global variables by including a synopsis and a description.

5.1 po_errno

```
extern int32 po_errno;
```

po_errno works just like errno. When an error occurs, the reason is left in po_errno.

5.2 po_errlist

```
extern char *po_errlist[];
```

po_errlist works just like sys_errlist[]. When an error occurs, a string description of the error can be found in po_errlist[po_errno].

5.3 po_real_time_world_state

```
extern ObjectID po_real_time_world_state;
```

po_real_time_world_state has the following components:

- po_real_time_world_state.simulator.site == 0
- po_real_time_world_state.simulator.host == 0
- po_real_time_world_state.object == 0

6 Functions

The sections below describe the libpo functions by including a synopsis and a description.

6.1 po_create

```
PO_DATABASE *po_create(db_type, exercise_id, database_id, sim_addr,
                       unit_db_version, terrain,
                       hostname[], simulator_type,
                       send_handler, send_user_data)
    int32              db_type;
    uint8              exercise_id;
    uint8              database_id;
    SimulationAddress *sim_addr;
    int16              unit_db_version;
    TerrainDatabaseID *terrain;
    char               hostname[];
    SimulatorType      simulator_type;
    SEND_HANDLER       send_handler;
    PO_USER_DATA_TYPE  send_user_data;
```

po_create creates a persistent object database.

The db_type is either PO_DATABASE_TYPE_PASSIVE or PO_DATABASE_TYPE_ACTIVE. simulator_type identifies the type of simulator that is opening the database (see Section 9.1.2 [Active Simulator], page 49), and thus implicitly indicates to other simulators that are maintaining this database what resources and services this simulator provides. The send_handler, which must be non-NULL, is used by libpo to send packets on the network. When a database is created, many libcallback events are created and stored in the returned database object. These events can be used to attach event handlers to. For descriptions of the event handlers, See Chapter 4 [Events and Event Handlers], page 7.

The returned PO_DATABASE should be passed to all libpo routines.

A NULL return value indicates an error occurred.

6.2 po_destroy

```
void po_destroy(db, cleanup)
    PO_DATABASE *db;
    int32       cleanup;
```

po_destroy closes out a database. This function *should always be called* when an application exits to ensure orderly transition of that application's objects.

The cleanup flag indicates whether allocated memory should be deallocated. This might take some time for a large database, and is generally unnecessary in unix if a program is exiting.

This function returns no value.

6.3 po_delete_all

```
int32 po_delete_all(db)
    PO_DATABASE *db;
```

po_delete_all completely initializes the shared database. This is a highly destructive function, and should be used with great caution.

The following handlers (if registered) can be invoked before this routine returns:

- entry_gone_event_handler (see Section 4.8 [entry'gone'event'handler], page 10)
- new_object_event_handler (see Section 4.9 [new'object'event'handler], page 10)
- object_changed_event_handler (see Section 4.10 [object'changed'event'handler], page 11)
- object_gone_event_handler (see Section 4.11 [object'gone'event'handler], page 11)

A NULL return value indicates an error occured.

6.4 po_process_packet

```
int32 po_process_packet(db, pdu, pdu_size)
    PO_DATABASE         *db;
    PersistentObjectPDU *pdu;
    int32                pdu_size;
```

`po_process_packet` is the dispatching packet handler for libpo. Pass it all `persistentObjectProtocolNumber` packets received from the network. Do not loop back locally sent packets to this function.

A `NULL` return value indicates an error occured.

6.5 po_tick

```
void po_tick(db)
    PO_DATABASE *db;
```

`po_tick` must be called at least once every 10 seconds to ensure retransmission and timeouts work correctly. However, the more frequently this routine is called, the more evenly distributed packet traffic will be, and hence, the better the network will perform. This routine does no searching, and hence one call per frame should cause no performance problems.

This function returns no value.

6.6 po_create_object

```
PO_DB_ENTRY *po_create_object(db, world_state, class,
                              missing_from_world_state,
                              variants, variant_size,
                              object_user_data)
    PO_DATABASE       *db;
    ObjectID          *world_state;
    uint8              class;
    uint32             missing_from_world_state;
    char              *variants;
    uint32             variant_size;
    PO_USER_DATA_TYPE object_user_data;
```

`po_create_object` creates a new object in the specified `world_state` (see Section 9.1.4 [World State], page 49). `world_state` can be `NULL`, in which case the object will be created in the current world state for the database. Do not create world states using this routine (call `po_create_world_state` instead (see Section 6.7 [po˘create˘world˘state], page 22)). Also, do not create `ExerciseInitializerClass` objects using this routine (call `po_set_exercise_initialization` (see Section 6.39 [po˘set˘exercise˘initialization], page 39).

The variants part is a pointer to a structure of type: LineClass, OverlayClass, PointClass, etc. The variant_size can be found using the following p_size.h macros:

- PRO_PO_WORLD_STATE_CLASS_SIZE
- PRO_PO_OVERLAY_CLASS_SIZE
- PRO_PO_POINT_CLASS_SIZE
- PRO_PO_LINE_CLASS_SIZE
- PRO_PO_SECTOR_CLASS_SIZE
- PRO_PO_TEXT_CLASS_SIZE
- PRO_PO_UNIT_CLASS_SIZE
- PRO_PO_HHOUR_CLASS_SIZE
- PRO_PO_STEALTH_CONTROLLER_SIZE

A world_state of NULL will lead to the object being created in the current world state (see Section 9.1.4 [World State], page 49) of the database as set by po_set_world_state.

The object_user_data will be assigned to the new object before any handlers are called. An application can use this field to identify whether objects were created locally or remotely (remote objects are guaranteed to have NULL object_user_data at create time).

The returned PO_DB_ENTRY is the unique handle to the created entry. It is needed to delete or change the entry.

The following handlers (if registered) can be invoked before this routine returns:

- new_entry_event_handler (see Section 4.6 [new'entry'event'handler], page 9)
- new_object_event_handler (see Section 4.9 [new'object'event'handler], page 10)

A NULL return value indicates an error occured.

6.7 po_create_world_state

```
PO_DB_ENTRY *po_create_world_state(db, base_state,
                                   description, secondsSince1970)
    PO_DATABASE  *db;
    ObjectID     *base_state;
    char          description[];
```

```
uint32        secondsSince1970;
```

po_create_world_state creates a new world state (see Section 9.1.4 [World State], page 49), built upon the passed base_state and with the description and time specified.

The returned PO_DB_ENTRY is the unique handle to the created object. It is needed to delete or change the object.

The following handlers (if registered) can be invoked before this routine returns:

- new_entry_event_handler (see Section 4.6 [new`entry`event`handler], page 9)
- new_object_event_handler (see Section 4.9 [new`object`event`handler], page 10)

A NULL return value indicates an error occured.

6.8 po_change_object

```
PO_DB_ENTRY *po_change_object(db, entry, variants, variant_size)
    PO_DATABASE *db;
    PO_DB_ENTRY *entry;
    char        *variants;
    uint32       variant_size;
```

po_change_object attempts to modify the object described in the passed entry. It will create a new entry if the object has not yet been duplicated in the current world state (see Section 9.1.4 [World State], page 49) of the database as set by po_set_world_state. Do not change ExerciseInitializerClass objects using this routine (call po_set_exercise_initialization (see Section 6.39 [po`set`exercise`initialization], page 39).

The variants part is a pointer to a structure of type: WorldStateClass, OverlayClass, PointClass, etc. The variant_size can be found using the following p_size.h macros:

- PRO_PO_WORLD_STATE_CLASS_SIZE
- PRO_PO_OVERLAY_CLASS_SIZE
- PRO_PO_POINT_CLASS_SIZE
- PRO_PO_LINE_CLASS_SIZE
- PRO_PO_SECTOR_CLASS_SIZE

- PRO_PO_TEXT_CLASS_SIZE
- PRO_PO_UNIT_CLASS_SIZE
- PRO_PO_HHOUR_CLASS_SIZE
- PRO_PO_STEALTH_CONTROLLER_SIZE

The modified entry is returned.

The following handlers (if registered) can be invoked before this routine returns:

- new_entry_event_handler (see Section 4.6 [new'entry'event'handler], page 9)
- entry_changed_event_handler (see Section 4.7 [entry'changed'event'handler], page 10)
- new_object_event_handler (see Section 4.9 [new'object'event'handler], page 10)
- object_changed_event_handler (see Section 4.10 [object'changed'event'handler], page 11)

If another simulator changes this object at the same time,
the object_change_failed_event_handler (see Section 4.12 [object'change'failed'event'handler], page 12) may also be called soon after.

A NULL return value indicates an error occured.

6.9 po_change_object_missing_flag

```
PO_DB_ENTRY *po_change_object_missing_flag(db, entry,
                                          missing_from_world_state)
    PO_DATABASE *db;
    PO_DB_ENTRY *entry;
    uint32       missing_from_world_state;
```

po_change_object_missing_flag attempts to modify the missingFromWorldState flag of the object described in the passed entry with respect to the current world state of the database as set by po_set_world_state. It will create a new entry if the object has not yet been duplicated into this world state (see Section 9.1.4 [World State], page 49).

The modified entry is returned.

The following handlers (if registered) can be invoked before this routine returns:

- entry_changed_event_handler (see Section 4.7 [entry'changed'event'handler], page 10)
- new_entry_event_handler (see Section 4.6 [new'entry'event'handler], page 9)
- new_object_event_handler (see Section 4.9 [new'object'event'handler], page 10)
- object_changed_event_handler (see Section 4.10 [object'changed'event'handler], page 11)
- object_gone_event_handler (see Section 4.11 [object'gone'event'handler], page 11)

If another simulator changes this object at the same time, the object_change_failed_event_handler (see Section 4.12 [object'change'failed'event'handler], page 12) may also be called soon after.

A NULL return value indicates an error occured.

6.10 po_change_entry

```
PO_DB_ENTRY *po_change_entry(db, entry, variants, variant_size)
    PO_DATABASE *db;
    PO_DB_ENTRY *entry;
    char        *variants;
    uint32       variant_size;
```

po_change_entry attempts to modify the passed entry. Do not change ExerciseInitializerClass objects using this routine (call po_set_exercise_initialization (see Section 6.39 [po'set'exercise'initialization], page 39).

The variants part is a pointer to a structure of type: WorldStateClass, OverlayClass, PointClass, etc. The variant_size can be found using the following p_size.h macros:

- PRO_PO_WORLD_STATE_CLASS_SIZE
- PRO_PO_OVERLAY_CLASS_SIZE
- PRO_PO_POINT_CLASS_SIZE
- PRO_PO_LINE_CLASS_SIZE
- PRO_PO_SECTOR_CLASS_SIZE
- PRO_PO_TEXT_CLASS_SIZE
- PRO_PO_UNIT_CLASS_SIZE
- PRO_PO_HHOUR_CLASS_SIZE

- `PRO_PO_STEALTH_CONTROLLER_SIZE`

The modified entry is returned.

The following handlers (if registered) can be invoked before this routine returns:

- `entry_changed_event_handler` (see Section 4.7 [entry˙changed˙event˙handler], page 10)
- `new_object_event_handler` (see Section 4.9 [new˙object˙event˙handler], page 10)
- `object_gone_event_handler` (see Section 4.11 [object˙gone˙event˙handler], page 11)

If another simulator changes this object at the same time, the `object_change_failed_event_handler` (see Section 4.12 [object˙change˙failed˙event˙handler], page 12) may also be called soon after.

A `NULL` return value indicates an error occured.

6.11 po_change_entry_missing_flag

```
PO_DB_ENTRY *po_change_entry_missing_flag(db, entry, missing_from_world_state)
    PO_DATABASE *db;
    PO_DB_ENTRY *entry;
    uint32       missing_from_world_state;
```

`po_change_entry_missing_flag` attempts to modify the `missingFromWorldState` flag of the passed entry.

The modified entry is returned.

The following handlers can be invoked before this routine returns:

- `entry_changed_event_handler` (see Section 4.7 [entry˙changed˙event˙handler], page 10)
- `new_object_event_handler` (see Section 4.9 [new˙object˙event˙handler], page 10)
- `object_changed_event_handler` (see Section 4.10 [object˙changed˙event˙handler], page 11)
- `object_gone_event_handler` (see Section 4.11 [object˙gone˙event˙handler], page 11)

If another simulator changes this object at the same time, the `object_change_failed_event_handler` (see Section 4.12 [object change failed event handler], page 12) may also be called soon after.

A NULL return value indicates an error occured.

6.12 po_copy_object_into_ws

```
PO_DB_ENTRY *po_copy_object_into_ws(db, entry, into_world_state)
    PO_DATABASE *db;
    PO_DB_ENTRY *entry;
    ObjectID    *into_world_state;
```

`po_copy_object_into_ws` attempts to duplicate the information specified in the `entry` into a version of that entry in the specified `into_ws`. If the object described by the `entry` already exists in the specified world state (see Section 9.1.4 [World State], page 49), the existing `entry` will be modified, otherwise a new entry will be created.

The new or modified entry is returned.

The following handlers (if registered) may be invoked before this routine returns:

- `new_entry_event_handler` (see Section 4.6 [new entry event handler], page 9)
- `entry_changed_event_handler` (see Section 4.7 [entry changed event handler], page 10)
- `new_object_event_handler` (see Section 4.9 [new object event handler], page 10)
- `object_changed_event_handler` (see Section 4.10 [object changed event handler], page 11)
- `object_gone_event_handler` (see Section 4.11 [object gone event handler], page 11)

If another simulator changes this object at the same time, the `object_change_failed_event_handler` (see Section 4.12 [object change failed event handler], page 12) may also be called soon after.

A NULL return value indicates an error occured.

6.13 po_set_object_user_data

```
void po_set_object_user_data(db, entry, val)
    PO_DATABASE       *db;
    PO_DB_ENTRY       *entry;
    PO_USER_DATA_TYPE val;
```

po_set_object_user_data routine sets the object_user_data to the passed value for all entries describing this object. The object user data of any entry regarding the same object can be found in entry->object_user_data.

This function returns no value.

6.14 po_entry_owner

```
char *po_entry_owner(db, entry)
    PO_DATABASE *db;
    PO_DB_ENTRY *entry;
```

po_entry_owner returns a pointer to the name of the host which currently owns the entry. This can be used by an application object_change_failed_event_handler (*see Section 4.12 [object˙change˙failed˙event˙handler], page 12*) to identify which machine changed the object.

6.15 po_simulator_name

```
char *po_simulator_name(db, address)
    PO_DATABASE       *db;
    SimulationAddress *address;
```

po_simulator_name returns the name of the host with the specified address.

6.16 po_find_simulator

```
PO_DB_ENTRY *po_find_simulator(db, address)
    PO_DATABASE       *db;
    SimulationAddress *address;
```

po_find_simulator returns a PO_DB_ENTRY containing a simulatorPresentPDU corresponding to the host with the specified address. If there is no host with the specified address acting as an active user of the PO database, NULL will be returned.

6.17 po_delete_object

```
int po_delete_object(db, entry)
    PO_DATABASE *db;
    PO_DB_ENTRY *entry;
```

po_delete_object routine deletes the specified object from the database.

The following handlers (if registered) can be invoked before this routine returns:

- entry_gone_event_handler (see Section 4.8 [entry`gone`event`handler], page 10)
- new_object_event_handler (see Section 4.9 [new`object`event`handler], page 10)
- object_changed_event_handler (see Section 4.10 [object`changed`event`handler], page 11)
- object_gone_event_handler (see Section 4.11 [object`gone`event`handler], page 11)
- world_state_changed_event_handler (see Section 4.15 [world`state`changed`event`handler], page 13)

A NULL return value indicates an error occured.

6.18 po_delete_objects

```
int32 po_delete_objects(db, n_entries, entries)
    PO_DATABASE *db;
    int32       n_entries;
    PO_DB_ENTRY *entries[];
```

po_delete_objects deletes the specified objects from the database.

The following handlers (if registered) can be invoked before this routine returns:

- entry_gone_event_handler (see Section 4.8 [entry`gone`event`handler], page 10)
- new_object_event_handler (see Section 4.9 [new`object`event`handler], page 10)

- object_changed_event_handler (see Section 4.10 [object'changed'event'handler], page 11)
- object_gone_event_handler (see Section 4.11 [object'gone'event'handler], page 11)
- world_state_changed_event_handler (see Section 4.15 [world'state'changed'event'handler], page 13)

A NULL return value indicates an error occured.

6.19 po_delete_entry

```
int32 po_delete_entry(db, entry)
    PO_DATABASE *db;
    PO_DB_ENTRY *entry;
```

po_delete_entry deletes the specified entry from the database.

The following handlers (if registered) may be invoked before this routine returns:

- entry_gone_event_handler (see Section 4.8 [entry'gone'event'handler], page 10)
- new_object_event_handler (see Section 4.9 [new'object'event'handler], page 10)
- object_changed_event_handler (see Section 4.10 [object'changed'event'handler], page 11)
- object_gone_event_handler (see Section 4.11 [object'gone'event'handler], page 11)
- world_state_changed_event_handler (see Section 4.15 [world'state'changed'event'handler], page 13)

A NULL return value indicates an error occured.

6.20 po_delete_entries

```
int32 po_delete_entries(db, n_entries, entries)
    PO_DATABASE *db;
    int32       n_entries;
    PO_DB_ENTRY *entries[];
```

po_delete_entries deletes the specified entries from the database.

The following handlers (if registered) may be invoked before this routine returns:

- **entry_gone_event_handler** (see Section 4.8 [entry'gone'event'handler], page 10)
- **new_object_event_handler** (see Section 4.9 [new'object'event'handler], page 10)
- **object_changed_event_handler** (see Section 4.10 [object'changed'event'handler], page 11)
- **object_gone_event_handler** (see Section 4.11 [object'gone'event'handler], page 11)
- **world_state_changed_event_handler** (see Section 4.15 [world'state'changed'event'handler], page 13)

A NULL return value indicates an error occured.

6.21 po_get_entry

```
PO_DB_ENTRY *po_get_entry(db, object_id, world_state_id)
    PO_DATABASE *db;
    ObjectID    *object_id;
    ObjectID    *world_state_id;
```

po_get_entry looks up the specified **objectID/worldStateID** pair and returns the associated entry.

A NULL return value indicates the entry was not found.

6.22 po_get_object

```
PO_DB_ENTRY *po_get_object(db, object_id)
    PO_DATABASE *db;
    ObjectID    *object_id;
```

po_get_object looks up the specified **objectID** in the current world state (see Section 9.1.4 [World State], page 49) of the database as set by **po_set_world_state** and returns the associated object.

A NULL return value indicates the object was not found.

6.23 po_query_for_current_objects

```
void po_query_for_current_objects(db, query_handler, query_user_data)
    PO_DATABASE      *db;
    PO_EVENT_HANDLER  query_handler;
    PO_USER_DATA_TYPE query_user_data;
```

`po_query_for_current_objects` invokes the `query_handler` on each object relevant to the current world state(see Section 9.1.4 [World State], page 49). Note that it *is not safe* to delete objects in the `query_handler` routine. Instead, an application should compile a list of objects to delete and delete them all with `po_delete_objects`.

This function returns no value.

6.24 po_query_for_all_entries

```
void po_query_for_all_entries(db, query_handler, query_user_data)
    PO_DATABASE      *db;
    PO_EVENT_HANDLER  query_handler;
    PO_USER_DATA_TYPE query_user_data;
```

`po_query_for_all_entries` invokes the `query_handler` on every entry with a pdu of type `describeObjectPDU`. Note that it *is not safe* to delete entries in the `query_handler` routine. Instead, an application should compile a list of entries to delete and delete them all with `po_delete_entries`.

T unction returns no value.

6.25 po_set_world_state

```
int32 po_set_world_state(db, entry)
    PO_DATABASE *db;
    PO_DB_ENTRY *entry;
```

`po_set_world_state` attempts to set the current world state (see Section 9.1.4 [World State], page 49) of the database to that described in the passed `entry`. Passing `NULL` as the `entry` indicates the Real Time world state.

The default world state is the Real Time world state. The current world state entry can be found in db->current_world_state (which will be NULL if in the Real Time world state). The objectID of the current world state can be found in db->current_world_state_id.

The following handlers (if registered) can be invoked before this routine returns:

- new_object_event_handler (see Section 4.9 [new'object'event'handler], page 10)
- object_changed_event_handler (see Section 4.10 [object'changed'event'handler], page 11)
- object_gone_event_handler (see Section 4.11 [object'gone'event'handler], page 11)
- world_state_changed_event_handler (see Section 4.15 [world'state'changed'event'handler], page 13)

A NULL return value indicates an error occured.

6.26 po_start_network_animation

```
int32 po_start_network_animation(db, world_state,
                                 secondsSince1970, clock_rate)
    PO_DATABASE *db;
    ObjectID    *world_state;
    uint32       secondsSince1970;
    float32      clock_rate;
```

po_start_network_animation sends a SetWorldStatePDU onto the network with the specified data, and keeps sending the PDU periodically until explicitly stopped via po_stop_network_animation.

A NULL return value indicates an error occured.

6.27 po_stop_network_animation

```
int32 po_stop_network_animation(db, world_state, secondsSince1970)
    PO_DATABASE *db;
    ObjectID    *world_state;
    uint32       secondsSince1970;
```

`po_stop_network_animation` sends one last `SetWorldStatePDU` with the specified `world_state` (see Section 9.1.4 [World State], page 49) and `time` and a clock rate of 0.0, and stops the periodic retransmission of the last `SetWorldStatePDU`.

A `NULL` return value indicates an error occured.

6.28 po_find_base_world_state

```
PO_DB_ENTRY *po_find_base_world_state(db, entry, skip_entry)
     PO_DATABASE *db;
     PO_DB_ENTRY *entry;
     PO_DB_ENTRY *skip_entry;
```

`po_find_base_world_state` is a convenience function which attempts to locate the world state (see Section 9.1.4 [World State], page 49) on which the passed world state `entry` is based. If non-null, entries will be checked against `skip_entry` in the search (this is provided as a convenient way to deal with deletion), and will not be chosen if they match.

A `NULL` return value indicates no base state was found.

6.29 po_find_overlay

```
PO_DB_ENTRY *po_find_overlay(db, entry)
     PO_DATABASE *db;
     PO_DB_ENTRY *entry;
```

`po_find_overlay` is a convenience function which attempts to locate the overlay to which an `entry` belongs.

A `NULL` return value indicates the overlay was not found.

6.30 po_object_color

```
uint8 po_object_color(db, entry)
     PO_DATABASE *db;
     PO_DB_ENTRY *entry;
```

po_object_color is a convenience function which returns the correct color code for the passed entry. It looks up the overlay of the entry if the entry color is OCOverlayDefault.

A NULL return value indicates the overlay was not found.

6.31 po_clear_change_flags

```
void po_clear_change_flags(unit)
    UnitClass *unit;
```

po_clear_change_flags is a convenience function which clears all the change flags in a unit class object (changeLocation, changeDirection, etc.).

6.32 po_save_all

```
int32 po_save_all(db, fname, save_scratch, save_realtime,
                  save_non_realtime)
    PO_DATABASE *db;
    char        *fname;
    int32        save_scratch;
    int32        save_realtime;
    int32        save_non_realtime;
```

po_save_all saves all world states, overlays, and other objects into a file named fname. If save_scratch is FALSE, scratch overlays and their members will not be saved. If save_realtime is FALSE, objects which have a world state (see Section 9.1.4 [World State], page 49) (i.e., anything other than overlays and world states) which are in the Real Time world state will not be saved. If save_non_realtime is FALSE, objects which have a world state which are not in the Real Time world state will not be saved. Setting save_realtime to FALSE is useful for saving only Courses of Action (see libcoa). Setting save_non_realtime to FALSE is useful for saving a scenario containing multiple overlays which exists only in the Real Time world state. po_save_all returns the number of objects saved.

A NULL return value indicates an error occured.

6.33 po_save_overlay

```
int32 po_save_overlay(db, fname, overlay, class_mask)
    PO_DATABASE *db;
    char        *fname;
    PO_DB_ENTRY *overlay;
    uint32       class_mask;
```

`po_save_overlay` saves the current world state (see Section 9.1.4 [World State], page 49) of all objects with classes indicated by `class_mask` into a file named `fname`, provided those objects are in the passed `overlay`, or are associated with an object in the passed `overlay`. `po_save_overlay` returns the number of objects saved.

The `class_mask` is a bitmask which indicates which object classes should be saved. For example, to save only points and lines, use:

```
(PO_CLASS_MASK(objectClassPoint) | PO_CLASS_MASK(objectClassLine))
```

`PO_FULL_CLASS_MASK` will allow all classes to be saved (except world states and other overlays, which are automatically eliminated).

A `NULL` return value indicates an error occured.

6.34 po_load_file

```
int32 po_load_file(db, fname,
                   overlay_confirmation_handler,
                   overlay_confirmation_user_data)
    PO_DATABASE                    *db;
    char                           *fname;
    PO_OVERLAY_CONFIRMATION_HANDLER overlay_confirmation_handler;
    PO_USER_DATA_TYPE               overlay_confirmation_user_data;
```

`po_load_file` loads the file named `fname`. Each overlay in the file is checked with the `overlay_confirmation_handler` to confirm that the application can accept it. If any errors occur during the load (including a failed `overlay_confirmation`) no objects will be created. `po_load_file` returns the number of objects loaded.

If the file was created with `po_save_all`, new world states (see Section 9.1.4 [World State], page 49) will be created to replicate the ones saved, and objects will be put into those world states (objects saved from the Real Time world state will be merged into the Real Time world state at

load time). If the file was created with po_save_overlay, the objects will be merged into the current world state of the database as set by po_set_world_state.

The following handlers (if registered) can be invoked before this routine returns:

- new_entry_event_handler (see Section 4.6 [new`entry`event`handler], page 9)
- entry_changed_event_handler (see Section 4.7 [entry`changed`event`handler], page 10)
- new_object_event_handler (see Section 4.9 [new`object`event`handler], page 10)
- object_changed_event_handler (see Section 4.10 [object`changed`event`handler], page 11)
- object_gone_event_handler (see Section 4.11 [object`gone`event`handler], page 11)

A NULL return value indicates an error occured.

6.35 po_new_stealth

```
void po_new_stealth(db, vehicleID)
    PO_DATABASE *db;
    ObjectID    *vehicleID;
```

An application should call po_new_stealth whenever a new Stealth vehicle appears in the SIMNET Stealth Protocol. libpo will determine whether a new stealthControllerClass object is needed, and will create one if necessary.

This function returns no value.

6.36 po_control_stealth

```
po_control_stealth(db, vehicleID, target_simulator)
    PO_DATABASE *db;
    ObjectID    *vehicleID;
    PO_DB_ENTRY *target_simulator;
```

po_control_stealth modifies the stealthControllerClass object regarding the identified stealth vehicle. It sets the controller field to the address of the target host. If target_simulator is NULL, the address used is that of the local host.

This function returns no value.

The following handler (if registered) can be invoked before this routine returns:

- `project_event_handler` (see Section 4.18 [project`event`handler], page 15)

6.37 po_time

```
uint32 po_time(db)
    PO_DATABASE *db;
```

`po_time` returns the current millisecond clock time of the shared database. Successive calls to `po_time` are *not* guaranteed to grow larger, and this clock *is not* an accurate measure of elapsed time. However, simultaneous calls to `po_time` on different simulators should yield approximately the same result (the master will lead the slaves by an amount equivalent to minimum network latency). It is appropriate to use `po_time` for comparisons with objects of class `HHour`.

6.38 po_get_exercise_initialization

```
ExerciseInitializerClass *po_get_exercise_initialization(db)
    PO_DATABASE *db;
```

`po_get_exercise_initialization` returns the current exercise initialization information as contained in the object of the `ExerciseInitializerClass`. In order that libpo guarantee that there is at most one object of this class, objects of this class should be created only by `po_set_exercise_initialization` (see Section 6.39 [po`set`exercise`initialization], page 39). If no object of this class exists in the database, a nominal value containing static default information will be returned.

6.39 po_set_exercise_initialization

```
void po_set_exercise_initialization(db, data)
    PO_DATABASE              *db;
    ExerciseInitializerClass *data;
```

po_set_exercise_initialization sets the current exercise initialization information by creating or changing the object of the **ExerciseInitializerClass**. In order that libpo guarantee that there is at most one object of this class, this function should solely be used to create or change this object. **po_create_object** (see Section 6.6 [po˙create˙object], page 21), **po_change_entry** (see Section 6.10 [po˙change˙entry], page 25), and **po_change_object** (see Section 6.8 [po˙change˙object], page 23) *should not* be used to manipulate objects of the **ExerciseInitializer** class.

po_set_exercise_initialization may be called only on active databases (see Section 9.1.2 [Active Simulator], page 49).

6.40 po_delete_exercise_initialization

```
void po_delete_exercise_initialization(db)
    PO_DATABASE              *db;
```

po_delete_exercise_initialization deletes the current exercise initialization information, if such information exists. This is performed by deleting the current object of the **ExerciseInitializerClass**, if it exists.

6.41 po_get_simulation_load

```
float32 po_get_simulation_load(db)
    PO_DATABASE              *db;
```

po_get_simulation_load returns the value of the current simulation load. Simulation load is an application defined quantity associated with all active (see Section 9.1.2 [Active Simulator], page 49) databases and corresponds with a simulator's quantity of simulationProtocol entities. Simulation load is set for local simulators via **po_set_simulation_load** (see Section 6.42 [po˙set˙simulation˙load], page 40) and can be retrieved for databases in remote simulators via the **simulationLoad** field of the **simulatorPresentPDUin** entries corresponding to simulators (see Section 4.4 [new˙simulator˙event˙handler], page 8). Simulation load is nominally between 0.0 and 1.0, with 1.0 representing maximum rated load. An overloaded simulator will have a simulation load greater than 1.0.

6.42 po_set_simulation_load

```
void po_set_simulation_load(db, val)
    PO_DATABASE          *db;
    float32               val;
```

po_set_simulation_load sets the value of the current simulation load for the passed database. See Section 6.41 [po`get`simulation`load], page 39 for a definition of simulation load.

6.43 po_least_simulation_loaded_simulator

```
PO_DB_ENTRY *po_least_simulation_loaded_simulator(db, sim_type, excluded_sim)
    PO_DATABASE          *db;
    SimulatorType         sim_type;
    SimulationAddress    *excluded_sim;
```

po_least_simulation_loaded_simulator returns an entry containing a simulatorPresentPDU corresponding to the simulator of the given SimulatorType that has the lowest simulation load (see Section 6.41 [po`get`simulation`load], page 39 for the definition of simulation load). If excluded_sim is not NULL, the simulator corresponding to that simulation address will be excluded from consideration. It is likely, but not guaranteed, that all active (see Section 9.1.2 [Active Simulator], page 49) *simulators in an exercise will agree on* what simulator is least loaded at a given instant in time.

7 Macros

The sections below describe the libpo macros by including a synopsis and a description.

7.1 PO_CLASS_MASK

```
uint32 PO_CLASS_MASK(objectClass)
    PersistentObjectClass objectClass;
```

PO_CLASS_MASK generates a bit mask to select the passed objectClass for calls to the function, po_save_overlay.

7.2 PO_FULL_CLASS_MASK

```
uint32 PO_FULL_CLASS_MASK
```

PO_FULL_CLASS_MASK generates a bit mask to select all object classes for calls to the function, po_save_overlay.

7.3 PO_OBJECT_DESCRIBE

```
DescribeObjectVariant PO_OBJECT_DESCRIBE
```

PO_OBJECT_DESCRIBE extracts the objects DescribeObjectVariant for the entry's most recent DescribeObjectPDU.

7.4 PO_OBJECT_CLASS

```
unsigned char PO_OBJECT_CLASS(entry)
    PO_DB_ENTRY *entry;
```

PO_OBJECT_CLASS extracts the object class from the entry's pdu.

7.5 PO_OBJECT_ID

```
ObjectID PO_OBJECT_ID(entry)
    PO_DB_ENTRY *entry;
```

`PO_OBJECT_ID` extracts the objectID from the entry's pdu.

7.6 PO_WORLD_STATE_DATA

```
WorldStateClass PO_WORLD_STATE_DATA(entry)
    PO_DB_ENTRY *entry;
```

`PO_WORLD_STATE_DATA` extracts the `WorldStateClass` variant from the entry's PDU.

7.7 PO_OVERLAY_DATA

```
OverlayClass PO_OVERLAY_DATA(entry)
    PO_DB_ENTRY *entry;
```

`PO_OVERLAY_DATA` extracts the `OverlayClass` variant from the entry's PDU.

7.8 PO_POINT_DATA

```
PointClass PO_POINT_DATA(entry)
    PO_DB_ENTRY *entry;
```

`PO_POINT_DATA` extracts the `PointClass` variant from the entry's PDU.

7.9 PO_LINE_DATA

```
LineClass PO_LINE_DATA(entry)
    PO_DB_ENTRY *entry;
```

`PO_LINE_DATA` extracts the `LineClass` variant from the entry's PDU.

7.10 PO_SECTOR_DATA

```
SectorClass PO_SECTOR_DATA(entry)
    PO_DB_ENTRY *entry;
```

PO_SECTOR_DATA extracts the SectorClass variant from the entry's PDU.

7.11 PO_TEXT_DATA

```
TextClass PO_TEXT_DATA(entry)
    PO_DB_ENTRY *entry;
```

PO_TEXT_DATA extracts the TextClass variant from the entry's PDU.

7.12 PO_UNIT_DATA

```
UnitClass PO_UNIT_DATA(entry)
    PO_DB_ENTRY *entry;
```

PO_UNIT_DATA extracts the UnitClass variant from the entry's PDU.

7.13 PO_HHOUR_DATA

```
HHourClass PO_HHOUR_DATA(entry)
    PO_DB_ENTRY *entry;
```

PO_HHOUR_DATA extracts the HHourClass variant from the entry's PDU.

7.14 PO_STEALTH_CONTROLLER_DATA

```
StealthControllerClass PO_STEALTH_CONTROLLER_DATA(entry)
    PO_DB_ENTRY *entry;
```

PO_STEALTH_CONTROLLER_DATA extracts the StealthControllerClass variant from the entry's PDU.

7.15 PO_TASK_DATA

```
TaskClass PO_TASK_DATA(entry)
    PO_DB_ENTRY *entry;
```

PO_TASK_DATA extracts the TaskClass variant from the entry's PDU.

7.16 PO_TASK_FRAME_DATA

```
TaskFrameClass PO_TASK_FRAME_DATA(entry)
    PO_DB_ENTRY *entry;
```

PO_TASK_FRAME_DATA extracts the TaskFrameClass variant from the entry's PDU.

7.17 PO_PARAMETRIC_INPUT_DATA

```
ParametricInputClass PO_PARAMETRIC_INPUT_DATA(entry)
    PO_DB_ENTRY *entry;
```

PO_PARAMETRIC_INPUT_DATA extracts the ParametricInputClass variant from the entry's PDU.

7.18 PO_PARAMETRIC_INPUT_HOLDER_DATA

```
ParametricInputHolderClass PO_PARAMETRIC_INPUT_HOLDER_DATA
    PO_DB_ENTRY *entry;
```

PO_PARAMETRIC_INPUT_HOLDER_DATA extracts the ParametricInputHolderClass variant from the entry's PDU.

7.19 PO_EXERCISE_INITIALIZER_DATA

```
ExerciseInitializerClass PO_EXERCISE_INITIALIZER_DATA
    PO_DB_ENTRY *entry;
```

PO_EXERCISE_INITIALIZER_DATA extracts the **ExerciseInitializerClass** variant from the entry's PDU.

8 Using Xtest

The test program xtest creates four windows, each representing a host on a simulated network. The simulated network has latency and errors just like a real network (although the simulated network makes more mistakes, to help find problems with libpo). Error messages are output to stdout of the controlling terminal.

There are the following Control Buttons:

Network: This toggle button attaches and detaches the simulated network cable from this simulated host.

New Overlay:
: Click left on this button to create a new overlay. Overlays have names like <#> and cycle through the 5 overlay colors.

COA: Toggle the COA button to post/unpost the Course Of Action editor. Using the COA editor, you can create new world states (see Section 9.1.4 [World State], page 49), delete world states, set the world state of the simulated host and do animation (either privately or on the simulated network). Full documentation of the COA interface can be found in libcoa/README.

Save All: Click left on this button to save everything in the shared database into a file called "all".

Load All: Click left on this button to load the file called "all" into the shared database.

Load Overlay:
: Click left on this button to load the file called "overlay" into the shared database.

Delete All:
: Click left on this button to delete everything in the shared database.

Quit: Click left on this button to exit this host's simulation. To stop xtest, hit ^C in its controlling window.

There are the following Display Areas:

Objects: At the top of each host display is a space where created objects are placed (the objects are class text with randomly chosen positions and sequentially chosen text like letter). Clicking left on an object changes it with respect to the current world state of the host. Clicking middle on an object removes it from the current world state of the host. Clicking right on an object deletes it entirely.

Overlays: Listed here are all the overlays in the system. Clicking left on one of these overlay buttons creates a new object in that overlay. Clicking middle on an overlay saves it in a file called "overlay". Clicking right on an overlay button deletes that overlay (and everything in it).

Known Simulators:
Listed here are all the known active simulators (see Section 9.1.2 [Active Simulator], page 49) on the simulated network.

Info: Below the simulators is the information display. Sliding the mouse pointer over any object updates this display with the objectID, worldStateID, sequence number, and owner of the object.

The following are the recommended X resources:

- POTest*foreground: Black
- POTest*background: Silver
- POTest*fontList: *helvetica-medium-r-normal–18*
- POTest*XmText*traversalOn: True
- POTest*traversalOn: False
- POTest*highlightOnEnter: False
- POTest*highlightThickness: 0

9 Protocol Specification

This section describes the Persistent Object Protocol. The information in this section is directly used by automatic scripts to generate DRN files which can generate machine readable and compilable descriptions of PDUs.

9.1 Terms

The following terms are essential for understanding the specifics of the Persistent Object Protocol:

9.1.1 Simulator

A *Simulator* is any machine on the network which handles packets (such as a workstation).

9.1.2 Active Simulator

An *Active Simulator* is a machine which is an active user of the Persistent Object Protocol. An active simulator may change the state of any Persistent Object. Active simulators have certain other responsibilities, including taking over object for simulators which go down. Active simulators are required to maintain a complete database, even if some object classes are always ignored by the simulator's application.

9.1.3 Passive Simulator

A *Passive Simulator* is a machine which is a passive observer of the Persistent Object Protocol. A passive simulator may not directly change the state of any Persistent Object. A passive simulator may keep an incomplete database including only those objects its application finds interesting.

9.1.4 World State

A *World State* is a snapshot of a set of objects at a particular time. World states have the following characteristics:

- To reduce storage requirements, world states are represented by a base state augmented by a series of deltas. Hence, when a new world state is created, it includes an ordered reference to all previous states upon which it was built.
- In order to display a particular world state, a search is necessary to find the correct representation for each object in the world.
- A world state may depend on any other world states which occur before or after it in time. In this way alternate futures and histories can be developed from one base state.
- Many world states may exist for a particular time.
- A new world state may exist on its own, depending on no other world states. Such world states should not be created unless absolutely necessary, since they lead to a rapid proliferation in the number of objects on the network.

9.1.5 Valid

An object is *Valid* only if all components of that object are reasonable. An line in an unknown overlay, for example, is invalid. The rules for dealing with invalid objects are explained for each object class.

9.2 Protocol Requirements

The OBG workstations require an environment in which objects can be shared and are robust. A protocol to support this environment must fulfill the following requirements:

- The number of objects in the system must be variable with a very high limit (in the thousands).
- The protocol used to communicate objects must work as a SIMNET protocol family, despite high packet loss rates.
- If at all possible, a simulator crash should not lead to the loss of the objects created on that simulator.
- Any simulator should be able to modify or delete any object.
- The protocol should be capable of representing different possible World States (see Section 9.1.4 [World State], page 49) simultaneously.
- Any object represented with this protocol must be completely transparent (i.e., it must have no state information which is not represented or derivable from the information in protocol packets).

The protocol used to share overlays between OBG workstations must fulfill the following requirements:

- Items on an overlay include points, lines, possibly other types of graphics, unit symbols.
- Task organization between unit symbols on an overlay must be represented in a machine-readable format.
- Enough information (such as a formation template) must be specified with each unit up to battalion echelon to represent its vehicles via SIMNET appearance PDUs.
- Enough information (such as a specific SIMNET echelon object type) must be specified with each unit up to battalion echelon to replicate it in a SAF simulation.

9.3 Protocol Definition

The Persistent Object Protocol is an application layer protocol between the SIMNET Association layer and a small sub-protocol which describes the classes and content of objects. This sub-protocol is transmitted as the data part of Describe Object PDUs. Describe Object PDUs also have a header which is used by the Persistent Object layer to maintain its database of objects. The interface to this protocol will be a library similar in nature to the Association Layer library.

All PDUs in this protocol should be transmitted using the Association Layer Datagram Service.

The Persistent Object Protocol has an exercise identifier which works the same way as a Simulation Protocol exercise identifier. It is expected that a simulator which is supporting Simulation Protocol and Persistent Object Protocol simultaneously will use the same exercise ID for both.

In addition, there is a database identifier field which can be used to further subdivide an exercise into several independent PO databases. This can be used, for example, to keep separate databases for SAF command and control, dynamic terrain, and environmental information. Each logical database has a unique space of Object IDs, so no references can be made between objects in separate databases.

The DRN definition of the top-level PDU can be found at the end of this specification.

NOTE: All maximum sizes assume a maximum datagram size of 2040 bytes.

9.3.1 Simulator Present PDU

Each simulator which is on the network which interacts with other simulators using the Persistent Object Protocol will broadcast a Simulator Present PDU every 20 seconds. This PDU acts as a heartbeat indicating that the simulator is running. The load field is used to encourage load balancing if the simulator goes down.

The time field is used to provide a consistent relative time across all simulators, for the exclusive use of Persistent Objects. When a simulator gets a time from another simulator's Simulator Present PDU which is larger than the time currently known to the simulator, the receiving simulator should immediately adopt that time. This time can be used as a base for other times (such as H-hour). When a simulator increases the database sequence number, it should reset this time to zero. Upon transitioning to a higher database sequence number each simulator should reset its own time to zero as well.

The databaseSequenceNumber field is used to facilitate a "delete all" procedure. Each simulator identifies in its Simulator Present PDU the current sequence number of the database. All objects with database numbers less than this number are invalid (see Section 9.1.5 [Valid], page 50). Upon receipt of a *Simulator Present* PDU, *the receiving simulator should* check the sequence number field against its own. If the number is lower, the receiving simulator should immediately retransmit its own Simulator Present PDU to ensure that the errant simulator is aware of the correct number. If the number is higher, the receiving simulator must delete all objects from its database, and adopt the new number as the correct version; it should then transmit a new Simulator Present PDU in reply. Until a simulator hears otherwise, it should assume the correct sequence number is zero. A simulator should immediately adopt the database sequence number in the first Simulator Present or Describe Object PDU it hears, provided that number is greater than the number it currently believes. Thereafter, only Simulator Present PDUs should be used to inherit new database sequence numbers.

To perform a "delete all" procedure, the deleting simulator should increment its own database sequence number, and immediately transmit a Simulator Present PDU. This is a drastic operation which should not be taken lightly by applications. Password protection would be appropriate.

If no Simulator Present PDU is received for 48 seconds, the receiver with the lowest declared load will take ownership of all the objects currently owned by the missing simulator. To ensure some simulator will take ownership, those simulators not doing so will nominate the simulator which they believe to have the lowest load. Procedures to negotiate between contenders for ownership are described elsewhere in this document.

If a Simulator Present PDU describes a new simulator, the receiver should restart periodic Describe Object PDU transmission for all objects which it owns, as though all the objects had just been changed. This way, simulators joining an exercise late can learn the complete state of the database without having to issue large numbers of Object Request PDUs.

```
constant simulatorPresentTransmitTime 20        -- seconds
constant simulatorPresentTimeoutTime 48         -- seconds

-- Longest at BBN is 25 chars
constant maxSPHostnameLength 32

type SimulatorPresentVariant sequence {

    -- Identity of the simulator
    simulator                    SimulationAddress,

    -- Resources provided by the simulator
    simulatorType                SimulatorType,
                                 unused(16),

    -- Identity of the shared database
    databaseSequenceNumber       UnsignedInteger (32),

    -- Measure of simulator load =
    --  (number of PO packets transmitted since last SimulatorPresentPDU) +
    --  square(number of packets missed during that time)
    load                         UnsignedInteger (32),

    -- Simulation load is an application defined measurement of
    --   a simulator's load of simulated entities.  The measurement is
    --   represented as a floating point number nominally between 0.0
    --   and 1.0.  1.0 represents 'full' rated loading, but a simulator
    --   may decide to simulate more than the full rated loading, thus
    --   generating a load > 1.0. Typically, only simulators with
    --   (simulatorType == simulator_LL_SAFSIM) will use this
    --   to determine what simulators can and should simulate vehicles.
    simulationLoad               Float (32),

    -- Milliseconds from time of last databaseSequenceNumber change
    time                         UnsignedInteger (32),

    -- PO Protocol Packets sent since last SimulatorPresentPDU
    packetsSent                  UnsignedInteger (32),

    -- Miscellaneous information for application level use
    unitDatabaseVersion          UnsignedInteger (16),
                                 unused (16),
    terrain                      TerrainDatabaseID,
```

```
    -- NULL terminated hostname
    hostname                    array (maxSPHostnameLength) of
                                  UnsignedInteger (8)
}
```

9.3.2 Describe Object PDU

A persistent object has the attributes that once created, it should continue to exist until deleted, despite simulator failure or other catastrophic error. To create such an object, a simulator uses a Describe Object PDU. After the header of this PDU is a sub-protocol which is used to describe different classes of objects. This protocol is easily extensible to describe a great variety of things which need this persistent behavior.

A persistent object can exist in many different states simultaneously. The collection of objects in a particular state is called a World State (see Section 9.1.4 [World State], page 49). There is a special World State (with object ID 0/0/0) which is the Real Time state (the actual time associated with this state is arbitrary, it may be a SIMNET simulated time). The Real Time state is implicit and is not created by any simulator. This state does not depend on any other world state, and no other world state may depend on it. The Real Time state is provided as a convenience for applications which do not need to manage multiple world states. All other states are associated with a time which cannot be changed once the state is created. An application which is not interested in supporting multiple world states, may ignore all objects with state other than 0/0/0 at the application layer. All active simulators (see Section 9.1.2 [Active Simulator], page 49) are, however, required to maintain the information regarding every object in the same exercise and database ID.

An object is uniquely identified by the pairing of its object identifier and its world state identifier. Two objects which have the same object identifier but different world state identifiers represent the same thing at two different times or in two different futures. An object is not valid (see Section 9.1.5 [Valid], page 50) unless its world state is the Real Time world state, or some other world state known to the receiving simulator.

All the different states of an object are maintained on the network, and it is up to the simulator (really the user of the simulator) to decide which world state should be displayed. Changes to one world state of an object impact the state of that object for that world state and all subsequent world states for which no new state was recorded. If an application does not desire this behavior, it may replicate the current object in all world states built from this state before changing the base object.

Invalid objects (objects which refer to other objects, which are not known to exist) may appear on the network from time to time due to packet misordering, network latency, or missed packets. When a new object appears on the network, it should be checked for validity. If it is not valid, it may be ignored. However, if an object already in the database becomes invalid, it should not be removed (see Section 9.1.5 [Valid], page 50).

9.3.3 Creating a Persistent Object

A persistent object is created by transmitting a Describe Object PDU with a unique Object ID.

The simulator which creates a persistent object is the original owner of that object. The initial sequence number of an object is 1. The owner is responsible for transmitting that object once every 30 seconds. In addition, the object is transmitted whenever it is changed. The sequence number is incremented by the simulator whenever any information within the object is altered. A simulator should disregard information received from the network if the sequence number is lower than that currently stored in the simulator's memory.

After transmitting the same information regarding an object ten times (for five minutes), the simulator should stop transmitting Describe Object PDUs regarding it and instead include its object ID and sequence number in an Objects Present PDU. If there is a need to transmit a Describe Object PDU regarding the object for any reason thereafter, it should be removed from the Objects Present PDU.

9.3.4 Creating a World State

A world state (see Section 9.1.4 [World State], page 49) is one class of persistent object, therefore declaring or changing a world state is done exactly the same way any other persistent object is created or changed. However, for a world state to be meaningful, the objects which are considered important for that state must be identified. This is done by creating new objects with the same object IDs, but different world state IDs.

A simulator which creates a world state and knows of no other world states other than the Real Time world state (which always exists, even in the absence of objects), will have to duplicate every object which should be included, changing only the world state ID and resetting the sequence number to 1 (and changing the owner, of course).

When a simulator creats a new world state, it must also create new objects for the graphics and units which are considered relevant to that world state and which have changed state between that world state and the most recent world state known to the simulator.

9.3.5 Changing a Persistent Object

A persistent object is changed by transmitting a Describe Object PDU regarding that object with the new state information. Among the items in the Describe Object PDU header, only the owner, sequence number, the simulation flag, and the missing flag may be changed after creating an object. Other restriction may apply to the particular object classes, as well.

It is an applications responsibility to manage user interaction with objects (limiting access to object created on other simulators, for example). The Object ID of an object does identify its creator. Note, however, that since a simulator may go down, an application which prevents users from deleting objects created on other simulators may make it impossible to remove objects from the system.

If a simulator other than the current owner wishes to change an object or take over ownership to replace a down simulator, it can do so by changing the owner field in subsequent PDUs. The simulator should immediately transmit the new information (along with the new owner field) with an incremented sequence number, and then retransmit the new information once every 30 seconds thereafter.

If a simulator receives a PDU regarding an object which it does not own, it should ignore the PDU if the sequence number is less than that currently known, or if the sequence numbers are the same and the owners are the same. Otherwise, if the sequence number is greater than that currently known or the sequence numbers are the same and the owner has changed, it should take the new information.

If a simulator receives a PDU regarding an object which it currently owns the new information may or may not be used, as follows:

- If the received PDU has a sequence number higher than that currently known to the simulator, the simulator must relinquish ownership of the object by accepting the new information.
- If the received PDU has a sequence number equal to that currently known to the simulator, the simulator will compare its own simulator address magnitude (SAM = host << 16 | site) to that of the simulator claiming ownership, and if higher, will increment the sequence number and retransmit the currently stored information.

If no PDUs have been received regarding an object created on another simulator for 72 seconds, each simulator should delete that object. Procedures for deleting objects are detailed below.

The various classes of objects are described next, followed by the DRN representation of the Describe Object PDU.

9.3.6 Object Classes

Object Classes include descriptions of world state (see Section 9.1.4 [World State], page 49), overlays (which have a name, color, etc.), items on overlays such as points, lines, or unit symbols, and other things which need to persist and have no particular owner.

To create an object of any class, follow the procedure described above for the creation of persistent objects. Some classes of objects have restrictions regarding which fields may be changed after creation. It is the applications responsibility to prevent these fields from changing.

9.3.6.1 World State Class

A world state (see Section 9.1.4 [World State], page 49) is made up of a uniquely identified time, and the state of selected objects at that time. Included in a world state object is a text description, and the sequence of world states upon which this is based. An empty history implies the state is based only on an unspecified Real Time State. Only objects of the World State class may appear in the history array.

Only the description of a world state object may be changed after it is first created.

Deleting a world state requires that all objects within that state be deleted.

```
constant maxWorldStateDescriptionLength 256
constant maxWorldStateHistoryLength 190

type WorldStateClass sequence {

    -- NULL terminated description
    description                 array (maxWorldStateDescriptionLength) of
                                  UnsignedInteger (8),

    -- Time of this frame
```

```
    secondsSince1970         UnsignedInteger (32),

    historyCount             UnsignedInteger (8),
                             unused (24),
    history                  array (historyCount) of ObjectID
}
```

9.3.6.2 Overlay Class

An overlay has name and color attributes, and is used to group other objects together for display purposes. To simplify semantics, overlays may only be created in the real time world state (see Section 9.1.4 [World State], page 49). While this prevents having an overlay change color or name at a particular time, it is necessary to ensure that all objects within the same overlay share the same overlay-inherited information. While scratch overlays (scratch flag is TRUE) are shared on the network for persistence, an application should not display a scratch overlay created by another workstation (`objectID.simulator` identifies the creator of an object).

Each overlay is tagged with a force ID which can be used to selectively filter display of overlays belonging to the "other side."

Any attribute of an overlay may be changed after initial creation.

Deleting an overlay requires that all overlay objects within that overlay are deleted.

```
type OverlayColor enum (8) {
    OCOverlayDefault (0),              -- Not valid in an Overlay Class
    OCBlack (1),
    OCYellow (2),
    OCRed (3),
    OCGreen (4),
    OCBlue (5)
}

constant maxOverlayNameLength 22

type OverlayClass sequence {

    -- NULL terminated overlay name
    name                     array (maxOverlayNameLength) of
                               UnsignedInteger (8),
```

```
        -- Default color of objects on this overlay
        color                          OverlayColor,
        scratch                        Boolean,
                                       unused (7),

        -- Force associated with this overlay
        forceID                        ForceID,
                                       unused (8)
    }
```

9.3.6.3 Point Class

A point has style, color, location, and direction attributes as well as the overlay into which the point is grouped. If dashed is True, the style should be modified (if possible) to show the point in a dashed fashion (used by the military to indicate that the information is uncertain).

A point is not valid (see Section 9.1.5 [Valid], page 50) unless its overlay is known to the receiving simulator.

Any attribute of a point may be changed after initial creation, including its overlay.

```
    type PointStyle enum (8) {
        PSgeneral (1),
        PScoordinating (2),
        PScontact (3),
        PScontrol (4),
        PSfortification (5),
        PSTAI (6),                     -- Target Area of Interest
        PSNAI (7),                     -- Named Area of Interest
        PSdecision (8)
    }

    type PointLocation sequence {
        x                              Integer (32),
        y                              Integer (32)
    }

    type PointClass sequence {

        -- Overlay to which this point belongs
        overlayID                      ObjectID,

        style                          PointStyle,
```

```
    color                       OverlayColor,
    dashed                      Boolean,
                                unused (31),
    location                    PointLocation,
    -- Some points (such as a NAI) have an associated direction
    direction                   Angle

}
```

9.3.6.4 Line Class

A line has style, color, thickness, and width attributes as well as the overlay into which the line is grouped, the points that make up the line, a closed flag indicating that the first and last point should be connected, and specification of arrow heads for each end. The use of line width depends upon the style: a plain line with non-zero width should be drawn as two parallel lines width meters apart (the points specify the centerline between the two); minefields are the same, except minefield symbols should be drawn along the centerline; width may not be meaningful for some line styles. Thickness is used to control the thickness of each drawn line segment. If dashed is True, the style should be modified (if possible) to show the line in a dashed fashion (used by the military to indicate that the information is uncertain). If splined is True, the workstation should use a splining function to smooth the line (if possible). The route flag is a user interface convenience, provided to allow interfaces to distinguish between routes and other control measures. The munition and density fields are used to attach munitions to the line, for instance minefields. The units of density depend on the type of munition.

Each point in a line has an identifier unique to that line which a receiving entity can use to determine the nature of changes. When a point is moved, its identifier should not change. When a point is inserted, it should be given an identifier unique to the line. The identifier 0 is reserved, and should not be used. A point is either a discrete location, or a reference to a road segment in the terrain database. Road segments need a direction to indicate whether the road segment points should be interpreted first-to-last or last-to-first.

A line is not valid (see Section 9.1.5 [Valid], page 50) unless its overlay is known to the receiving simulator.

Any attribute of a line may be changed after initial creation, including its overlay.

```
type LineStyle enum (8) {
```

```
        LSplain (1),
        LSfrontA (2),
        LSfrontB (3),
        LSminefield (4),
        LSminefieldAT (5),
        LSminefieldAP (6),
        LSberm (7),
        LSATDitchA (8),
        LSATDitchB (9),
        LSfortification (10),
        LSwire (11)
    }

    type ArrowHeadStyle enum (8) {
        noArrowHead (0),
        lineArrowHead (1),
        blockArrowHead (2)
    }

    type Direction enum (8) {
        firstToLast (1),
        lastToFirst (2)
    }

    type RoadSegment sequence {

        index                       Integer (32),
        direction                   Direction,
        fake                        Boolean,
                                    unused (23)

    }

    type PointType enum (8) {
        PTRoadSegment (1),
        PTLocation (2)
    }

    type PointDescription sequence {

        pointNumber                 Integer (16),
        pointType                   PointType,
                                    unused (8),
        variant                     choice (pointType) of {

            when (PTRoadSegment)
              roadSegment           RoadSegment,

            when (PTLocation)
              location              PointLocation
```

```
    }
}

type LineClass sequence {

    -- Overlay to which this line belongs
    overlayID                   ObjectID,

    style                       LineStyle,
    color                       OverlayColor,
    pointCount                  UnsignedInteger (8),
    thickness                   UnsignedInteger (8),   -- Pixels
    width                       UnsignedInteger (16),  -- Meters
    beginArrowHead              ArrowHeadStyle,
    endArrowHead                ArrowHeadStyle,
    -- Indicates if first point should be linked to last
    closed                      Boolean,
    -- Indicates that line segments should be dashed
    dashed                      Boolean,
    -- Indicates that a splining function should be used between vertices
    splined                     Boolean,
    -- Indicates that the line is a route
    route                       Boolean,
                                unused (12),
    -- Indicates a munition attached to the line
    munition                    ObjectType,
    -- Indicates a density of the munition
    density                     Float (32),

    points                      array (pointCount) of PointDescription

}
```

9.3.6.5 Sector Class

A sector has style, color, and thickness attributes as well as an origin and various radii. Thickness is used to control the thickness of each drawn line segment. A sector originates at the specified origin, and consists of a portion of a pie-slice shape bounded by two lines. Each bounding line should be drawn from the minimum radius to the maximum radius (if the minimum radius and the maximum radius are the same, no bounding line are drawn). Arcs should be drawn at each of the radii specified in the array. These radii need not be within the bounds of the minimum and maximum radii. If dashed is True, the style should be modified (if possible) to show the sector in a dashed fashion (used by the military to indicate that the information is uncertain).

A sector is not valid (see Section 9.1.5 [Valid], page 50) unless its overlay is known to the receiving simulator.

Any attribute of a sector may be changed after initial creation, including its overlay.

```
type SectorStyle enum (8) {
    SSplain (1)
}

constant maxArcRadii 200

type SectorClass sequence {

    -- Overlay to which this sector belongs
    overlayID                  ObjectID,

    style                      SectorStyle,
    color                      OverlayColor,

    -- Origin of the arc
    origin                     PointLocation,
    -- Starting and ending points of the arc sides (0 means start at origin)
    minRadius                  UnsignedInteger (32),
    maxRadius                  UnsignedInteger (32),
    -- SIMNET style angle of counter-clockwise most side
    initialAngle               Angle,
    -- Magnitude of arc extent clockwise from initialAngle
    extent                     Angle,

    thickness                  Integer (8),    -- Pixels

    -- An arc should be drawn between the two sides at each listed radius
    radiusCount                UnsignedInteger (8),
    dashed                     Boolean,
                               unused (15),
    arcRadii                   array (radiusCount) of UnsignedInteger (32)

}
```

9.3.6.6 Text Class

Text has font, color, and location attributes as well as the overlay into which the text is grouped. The alignment field is needed to compensate for differences in font definitions between different simulators. The offset fields allow text to remain a fixed distance from the specified point, despite

map scaling. Rather than having fixed label fields included in other graphic object classes (point, line, etc.), each text item is represented individually. To label both ends of a line, for example, two text objects must be created. This scheme greatly reduces bandwidth requirements when there are many unlabeled graphics, and also allows different applications to handle labeling graphics differently. Text with a length of zero is allowed, but discouraged.

Text is not valid (see Section 9.1.5 [Valid], page 50) unless its overlay is known to the receiving simulator. Text with a non-zero associated object is valid only if that object is known to the workstation.

Any attribute of a point may be changed after initial creation, including its overlay, and its associated object.

Deleting text does not require deleting the associated object, however, deleting any other object does require deleting text associated with that object.

```
type TextSize enum (8) {
    tinyText   (0),
    smallText  (1),
    mediumText (2),
    largeText  (3),
    hugeText   (4)
}

-- example of northWest alignment:
--
-- TEXT
--     (location)

type TextAlignment enum (8) {
    northWest (1),
    north (2),
    northEast (3),
    west (4),
    center (5),
    east (6),
    southWest (7),
    south (8),
    southEast (9)
}

constant maxTextLength 1024

type TextClass sequence {
```

```
        -- Overlay to which this text belongs
        overlayID                  ObjectID,

        size                       TextSize,
        color                      OverlayColor,
        length                     Integer (16),   -- Includes NULL terminator
        -- Position of text relative to location
        alignment                  TextAlignment,
                                   unused (8),
        location                   PointLocation,
        horizontalOffset           Integer (16),   -- Pixels
        verticalOffset             Integer (16),   -- Pixels

        -- Associated Object or 0/0/0 if none
        associatedObject           ObjectID,
        -- If associatedObject is Line Class, identifies which point
        associatedPointNumber      Integer (16),

        -- NULL terminated text string
        text                       array (length) of Integer (8)

    }
```

9.3.6.7 Unit Class

A unit has many attributes. Object type is any valid SIMNET object type of any domain (echelons, vehicles, etc.). The force ID, appearance, and marking fields are interpreted as in Simulation Protocol Vehicle Appearance PDUs. The SIMNET object type uniquely identifies an echelon in the common unit database (each simulator must identify which revision of the unit database it is using in its Machine Present PDU to ensure compatibility). The formation template field identifies how the member vehicles of this unit should be placed if displayed. If subordinates represented is True, the application should not attempt to get subordinate information from the unit database regarding this object. Each object can have both an organic and an attachment parent unit. The location of the unit is its center of mass. The direction is the direction of the unit relative to the formation database. Also, every subordinate unit is assumed to be facing this direction unless specifically overridden with separate objects. The unit strength is an arbitrary measure in the range 0.0 to 1.0. The time date group indicates the time and date at which the information was first valid. This may be before or after the time of the world state (see Section 9.1.4 [World State], page 49) in which this object resides. If dashed is True, the unit symbol should be modified (if possible) to draw in a dashed fashion (used by the military to indicate that the information is uncertain). If dugIn is True, projections of the unit should be given a Z value somewhat lower than ground level (exactly how much depends upon the objectType).

A unit is not valid (see Section 9.1.5 [Valid], page 50) unless its overlay is known to the receiving simulator.

A task organization may transcend multiple overlays.

Any attribute of a unit may be changed after initial creation except object type and force ID.

Relationships between unit objects require a series of conventions to be followed by all applications.

- If any immediate subordinates of a unit are altered, objects representing all database derived immediate subordinates of that unit must be maintained. For example, adjusting the location of one platoon in a company requires that all other platoons and the company headquarters vehicles be created as objects, and that the subordinates represented flag of the company be set to True.
- Attachment information is considered independent of unit database inquiries. For example, attaching an individual vehicle to a platoon does not require that the vehicles in that platoon be represented, or that their locations be changed.
- Applications are responsible for the integrity of all objects impacted by a change to a related object. Center of mass, for example must be recalculated for all superior units when an inferior unit is changed.
- If any member of an organization is represented in a world state (see Section 9.1.4 [World State], page 49), all known members of that organization must be represented in that world state.

```
constant maxFormationNameLength 36
constant maxUnitMunitions        8

type TimeDateGroup sequence {
    time                     UnsignedInteger (32),    -- HHMMSS
    date                     UnsignedInteger (32),    -- YYMMDD
    secondsSince1970         UnsignedInteger (32)
}

type SAFMethodology enum (8) {
    SAFMethodFundamental (0),
    SAFMethodSoar (1)
}

type UnitClass sequence {
```

```
-- Address of workstation commanding this unit, if being simulated
commander                    SimulationAddress,
-- Address of host simulating this unit, if being simulated
simulator                    SimulationAddress,

-- If being simulated, vehicleID being used to simulate this unit
simulationID                 VehicleID,

-- Desired SAF simulation methodology
methodology                  SAFMethodology,

-- The following are flags which are set by the user interface to
-- request that a specific field change be applied to the simulated
-- unit.  Some fields which the simulation would not change (such as
-- marking) are always updated if the unit is changed, hence they do
-- not have a flag.  Similarly, fields which the user interface
-- would not modify (such as objectType) do not need flags.
changeFormation              Boolean,
changeLocation               Boolean,
changeDirection              Boolean,
changeUnitStrength           Boolean,
changeMunitions              Boolean,
changeAppearance             Boolean,

                             unused (2),

-- Overlay to which this unit belongs
overlayID                    ObjectID,
forceID                      ForceID,
subordinatesRepresented      Boolean,
dashed                       Boolean,
                             unused (1),
shouldBeSimulated            Boolean,
simulated                    Boolean,
                             unused (3),
objectType                   ObjectType,
marking                      VehicleMarking,

-- NULL terminated formation name
formationTemplate            array (maxFormationNameLength) of
                                     Integer (8),

-- ID of organic parent (0/0/0 if not represented)
organicTo                    ObjectID,
-- ID of attached parent (0/0/0 if same as organic)
attachedTo                   ObjectID,

-- State information
location                     WorldCoordinates, -- Center of Mass
direction                    Angle,            -- Orientation of Formation
```

```
        unitStrength          Float (32),       -- 0 <= unitStrength <= 1
        timeDateGroup         TimeDateGroup,    -- Valid TDG of data
        munitions             array (maxUnitMunitions) of
                                 MunitionQuantity,
        job                   UnsignedInteger (32),  -- App. specific job
        appearance            UnsignedInteger (32),  -- Appearance
        taskFrameStack        ObjectID,    -- Points to top task in a stack
        backgroundFrame       ObjectID,    -- Background task
        parameters            ObjectID     -- Point to ParametricInputHolder
    }
```

9.3.6.8 Commo Class

A Commo object is similar to a radio net. Commo objects are used to transmit messages between entities via the PO database.

The message portion of the commo should not be deleted.

```
    constant maxCommoMessageLength 1024
    constant maxCommoNameLength      32

    type CommoClass sequence {

        name                  array (maxCommoNameLength) of Integer (8),
        sender                ObjectID, -- or a VehicleID
                              unused (16),

        alignment             TextAlignment,

        class                 Integer (8),
        associatedObject      ObjectID,

        messageId             Integer (32),
        message               array (maxCommoMessageLength) of Integer (8)
    }
```

9.3.6.9 Stealth Controller Class

Stealth controller objects are used to manage control of available stealth vehicles. The ObjectID of the stealth controller object should be the VehicleID of the stealth, to prevent accidental creation

of more than one controller object. The controller field identifies the address of the machine which is controlling the stealth.

The controller field may be changed after initial creation.

Stealth controller objects should not be deleted. If the database sequence number increases, each simulator should replicate all stealth controller objects from the previous database to ensure the objects are not lost. Applications should create a new stealth controller object for each stealth heard in the SIMNET Stealth Protocol.

To take control of a stealth vehicle, an application sets the controller field of the appropriate stealth controller object to its own address. The controller is responsible for projecting Persistent Object Protocol objects in the SIMNET Simulation Protocol as Vehicle Appearance Packets. If a machine receives a stealth controller object which identifies it as the controller, it should project vehicles as though it had volunteered. In this way, the user of one workstation can see in 3D the objects being manipulated by another workstation.

```
type StealthControllerClass sequence {

    -- Address of the controller
    controller                  SimulationAddress

}
```

9.3.6.10 H-Hour Class

H-hour objects are used to create a relative time upon which other times may be based (e.g., execute this route at H-hour + 15). H-hour is measured against the Persistent Object Database clock maintained via Simulator Present PDUs. The defined flag indicates whether the H-hour has been set by a user, or is currently undefined. (An undefined H-hour is used by SAF to indicate that units should hold until the H-hour is specified.)

Any attribute of an H-hour may be changed after initial creation.

```
constant maxHHourNameLength 22

type HHourClass sequence {
```

```
        -- NULL terminated HHour name
        name                        array (maxHHourNameLength) of
                                      UnsignedInteger (8),

        -- Force which is using this HHour
        forceID                     ForceID,

        -- Is this H-hour set?
        defined                     Boolean,
                                    unused (7),

        -- Time in terms of PO time
        time                        UnsignedInteger(32)

    }
```

9.3.6.11 Task Class

A Task object represents an individual behavior of SAF vehicles or units, such as avoid collisions, go to point, or follow road. Task behaviors are implemented as finite state machines which operate on the state defined in the Task object. The unformatted data in the Task object contains the state and argument data for the task. This data is interpreted by the task state machine which implements the task. Task arguments which are references to other Persistent Objects must be represented in the references section.

Any attribute of a Task may be changed after initial creation except for the size of the task data.

A task is not valid (see Section 9.1.5 [Valid], page 50) unless its references are known to the receiving simulator.

```
    constant maxTaskPOReferences 16

    type TaskClass sequence {

        -- Model name of task
        model       UnsignedInteger (32),

        -- Frame that this task is in
        frame       ObjectID,
```

```
        -- Number of references to other POs
        refcount    UnsignedInteger (8),
                    unused (8),

        -- References to other PO's, (to be used as Task arguments)
        references  array (maxTaskPOReferences) of ObjectID,

        -- Size of data for this task
        size        UnsignedInteger (16),
                    unused (16),

        -- Data for this task
        data        array (size) of UnsignedInteger (8)

    }
```

9.3.6.12 Task Frame Class

A Task Frame object groups a collection of zero or more tasks which execute in parallel. Each Task Object within a logical frame points back to this object via its 'frame' attribute.

The name of the task frame is used only by the user interface, and for debugging. Frames are linked together via the mission link to form a mission tree. Frames are linked together via the stack link to form the stack of tasks being executed by a unit.

Certain Task Frames are opaque. When a task manager is assembling a list of tasks to run, all tasks for each frame in a stack are considered until reaching an opaque frame. Versions of the same task higher in the stack take precedence of those lower in the stack.

Task Frames are pushed onto vehicle stacks as follows. Only the vehicle representing the unit can push a frame onto the unit's stack. Another agent can request that a frame be pushed by creating the frame and specifying the unit in the frame. Upon receipt, the vehicle representing the unit may decide whether the task frame was received (if a radio model is being used), and it may decide whether to push the task frame onto its task frame stack.

Special tasks called Enabling Tasks live in the Task Frame. They have the feature that they are only run when their frame is NOT active. The task manager which distributes tasks to units should run each enabling task which belongs to a frame which has a 'previousMissionFrame' specified as a currently executing frame. Note that enabling tasks must not have private state, since multiple copies of the same task (with different parameters and public state) may be running on the same

vehicle. This should not be a problem, since enabling tasks are not state machines, but rather predicate functions. When the enabling task detects that its predicate is fulfilled, it is responsible for starting the frame in which it resides.

Since enabling tasks are pointed at by frames, an enabling task should not point to a frame (else you will get a circular PO structure). Hence, enabling tasks can be recognized by the fact that they have (0/0/0) in their frame pointer.

A logic Stack is used to implement postfix boolean logic operations on what enabling tasks must evaluate to true to activate this frame. This logic Stack is read starting at 0 and stops when the value taskFrameLogicStackSTOP is reached. Specific Enabling Tasks are referred to by small indexs, and boolean operations are referred to by the constants taskFrameLogicStackOR, taskFrameLogicStackAND, and taskFrameLogicStackNOT.

```
constant maxTaskFrameNameLength 32
constant maxTaskFrameEnablingTasks 32
constant taskFrameLogicStackSize 128
constant taskFrameLogicStackNOT  252
constant taskFrameLogicStackOR   253
constant taskFrameLogicStackAND  254
constant taskFrameLogicStackSTOP 255

type TaskInstallationInstruction enum (8) {

        TIIPopNone (0),       -- Just push this frame onto the stack
        TIIPopNonOpaque (1),  -- Pop all non-opaque frames down to the first
                              -- opaque frame, then push this frame
        TIIPopOpaque (2)      -- Pop all frames down to and including the
                              -- first opaque frame, then push this frame
}

type TaskFrameClass sequence {
    -- NULL terminated Task Frame name
    name                 array (maxTaskFrameNameLength) of
                           UnsignedInteger (8),

    -- Whether tasks in frames stacked below this are hidden by this frame
    opaque               Boolean,

    -- Whether this is a task frame which should be destroyed if the
    -- unit ever stops executing it
    destroyWhenDone      Boolean,

                         unused (6),
```

```
        -- What to do to the task stack when this frame is executed/installed
        instruction          TaskInstallationInstruction,

        -- Unit which is being requested to push this Frame onto it's stack
        unit                 ObjectID,

        -- Next pointer used to implement a Unit's Stack of Frames
        nextStackFrame       ObjectID,

        -- Previous Task Frame in the Mission
        previousMissionFrame ObjectID,

        -- Postfix stack of logic operations to combine enabling tasks
        logicStack          array (taskFrameLogicStackSize) of
                              UnsignedInteger (8),

        -- Tasks which are run when this frame is NOT active.  These
        --   tasks may cause this frame to be executed by a unit
        etaskCount          UnsignedInteger (8),
                            unused (8),
        enablingTask        array (etaskCount) of ObjectID
    }
```

9.3.6.13 Parametric Input Class

The Parametric Input Class is used to store parameters to a SAF model. A model is a functional subsystem of a vehicle simulation, such as target selection. Large blocks of parameters can be created by linking then together through the chain field. To ensure consistency, a chain is not valid unless all elements of the chain have the same chain serial number.

Any attribute of a Parametric Input object may be changed after initial creation.

```
    constant maxParametricInputClassSize 1024

    type ParametricInputClass sequence {

            -- Link to next PI Class object (0/0/0 if this class is complete)
            chain   ObjectID,

            -- Serial number common to all elements of a chain
            chainSerial UnsignedInteger (8),
                        unused (8),

            -- Size of the parametric data in this PI Class
```

```
            size     UnsignedInteger (16),
                     unused (16),
                     unused (32),

            -- Data
            data     array (size) of UnsignedInteger (8)
    }
```

9.3.6.14 Parametric Input Holder Class

A Parametric Input Holder Class contains a collection of Parametric Input objects. Each object is tagged with a SAFModel identifier which indicates for what model the parametric data refers to.

Any attribute of a Parametric Input Holder object may be changed after initial creation except for size.

```
    type TaggedParametricInput sequence {

            -- Model for which this parameter data is for
            model   UnsignedInteger(32),

            -- Pointer to a Parametric Input Object
            data    ObjectID
    }

    constant maxParametricInputHolderInputs 128

    type ParametricInputHolderClass sequence {

            -- Number of parameters in the holder
            size    UnsignedInteger(16),
                    unused (16),

            -- array of tagged parameters
            blocks  array(size) of TaggedParametricInput
    }
```

9.3.6.15 Exercise Object Class

An Exercise Initializer object is used to synchronize multiple simulators with the same exercise

information. It is only valid for one Exercise Initializer object to exist in the database at one time (libpo ensures this). When a simulator receives an Exercise Initializer object for the first time, or if it receives notification that this object has changed, the simulator should adjust itself to the exercise parameters such as terrain database and battle scheme.

The Exercise Initializer object also broadcasts information about the simulation rate (a number >= 0.0) that is in effect. If a simulator wants to change the simulation rate for all simulators in the exercise, it will change this object and set the rateTimeStart field to be some PO time (see Section 9.3.1 [Simulator Present PDU], page 52) in the future. The time chosen should be far enough into the future so that all simulators can find out about the change before the rate actually goes into effect.

Any attribute of an Exercise Initializer object may be changed after initial creation.

```
type ExerciseInitializerClass sequence {

        -- The terrain database chosen for the exercise:
        terrain         TerrainDatabaseID,

        -- The battle scheme chosen for the exercise:
        battleScheme    BattleScheme,
                        unused (24),

        -- The rate at which simulation should be running
        simulationRate  Float (32),

        -- The PO time when the simulationRate is valid
        rateTimeStart   UnsignedInteger(32)
}
```

9.3.6.16 Describe Object PDU Definition

```
constant describeObjectTransmitTime 30          -- seconds
constant describeObjectTimeoutTime 72           -- seconds
constant describeObjectRetransmitTime 300       -- seconds

type PersistentObjectClass enum (8) {
    objectClassWorldState (1),
    objectClassOverlay (2),
    objectClassPoint (3),
```

```
    objectClassLine (4),
    objectClassSector (5),
    objectClassText (6),
    objectClassUnit (7),
    objectClassStealthController (9),
    objectClassHHour (10),
    objectClassCommo (12),
    objectClassTask (13),
    objectClassTaskFrame (14),
    objectClassParametricInput (15),
    objectClassParametricInputHolder (16),
    objectClassExerciseInitializer (17)
}

type DescribeObjectVariant sequence {

    -- Identity of the shared database
    databaseSequenceNumber      UnsignedInteger (32),

    -- Identity of the object
    objectID                    ObjectID,

    -- World State to which this object belongs or 0/0/0 if in the
    -- Real Time World State
    worldStateID                ObjectID,

    -- Identity of the simulator which first currently takes
    -- responsibility for this object
    owner                       SimulationAddress,

    -- Sequence number of this revision of the Object
    sequenceNumber              UnsignedInteger (16),

    class                       PersistentObjectClass,

    -- If true, this object does not exist in this world state, however
    -- it does exist in other world states.
    missingFromWorldState       Boolean,
                                unused (7),

    variant                     choice (class) of {

        when (objectClassWorldState)
          worldState            WorldStateClass,

        when (objectClassOverlay)
          overlay               OverlayClass,

        when (objectClassPoint)
          point                 PointClass,
```

```
        when (objectClassLine)
          line                  LineClass,

        when (objectClassSector)
          sector                SectorClass,

        when (objectClassText)
          text                  TextClass,

        when (objectClassUnit)
          unit                  UnitClass,

        when (objectClassStealthController)
          stealthController     StealthControllerClass,

        when (objectClassHHour)
          hHour                 HHourClass,

        when (objectClassCommo)
          commo                 CommoClass,

        when (objectClassTask)
          task                  TaskClass,

        when (objectClassTaskFrame)
          taskFrame             TaskFrameClass,

        when (objectClassParametricInput)
          parametricInput       ParametricInputClass,

        when (objectClassParametricInputHolder)
          parametricInputHolder ParametricInputHolderClass,

        when (objectClassExerciseInitializer)
          exerciseInitializer   ExerciseInitializerClass

    }
  }
```

9.3.7 Objects Present PDU

After sending an unchanged Describe Object PDU for five minutes, a simulator should stop sending the full Describe Object PDU, and instead confirm that object's presence by including its objectID/worldStateID pair and sequence number in an Objects Present PDU. Each of these PDUs is transmitted once every 30 seconds.

Upon receipt of an Objects Present PDU, each simulator should check whether it knows of the objects included. If an object is known and the sequence number and owner are the same as that known, the simulator should reset the timeout counter for that object. The receiver should send an Object Request PDU identifying each object which does not meet this criteria.

As a space optimization, only one worldStateID is allowed per Objects Present PDU. Each objectID should be paired with this worldStateID to find its unique identifier.

```
constant objectsPresentTransmitTime 30          -- seconds
constant maxObjectsPresentCount 150

type ObjectIDSequencePair sequence {

    objectID                    ObjectID,
    sequenceNumber              UnsignedInteger (16)

}

type ObjectsPresentVariant sequence {

    owner                       SimulationAddress,

    worldStateID                ObjectID,

    objectCount                 UnsignedInteger (8),
                                unused (8),
    objects                     array (objectCount) of
                                  ObjectIDSequencePair

}
```

9.3.8 Object Request PDU

A simulator may issue an Object Request PDU in response to an Objects Present PDU for each object which is:

- unknown to the receiver,
- thought to be owned by a different simulator than that identified in the Objects Present PDU, or

- thought to have a lower sequence number than that given in the Objects Present PDU.

This way, in the unlikely event that a simulator missed all the normal retransmissions of a new or changed object, it can still find out the object's state.

Upon receipt of an Object Request PDU, the simulator identified as the owner should move the specified object out of its Object Present PDUs and restart the normal retransmission of the object as though the object were just changed.

A passive simulator (see Section 9.1.3 [Passive Simulator], page 49) should use this PDU to get information regarding objects which it has filtered, or at startup to learn about the database. Active simulators (see Section 9.1.2 [Active Simulator], page 49) should not send this PDU until they have been on the net for some time, since their simulator present PDUs will automatically trigger describe object PDU transmission for all objects.

This is the only PDU that a passive simulator is allowed to send.

```
constant maxObjectRequestCount 150

type ObjectRequestVariant sequence {

    requestingSimulator        SimulationAddress,

    objectOwner                SimulationAddress,

    worldStateID               ObjectID,

    objectCount                UnsignedInteger (8),
                               unused (8),
    objects                    array (objectCount) of ObjectID

}
```

9.3.9 Delete Objects PDU

To remove persistent objects, the simulator must issue a Delete Objects PDU. The deleting simulator should be prepared to rebroadcast the Delete Objects PDU in response to each Describe Object PDU received regarding one of the objects for a duration of 5 minutes.

Upon receipt of a Delete Objects PDU, each simulator should delete the objects. If a simulator is the owner if a deleted object, it should stop broadcasting PDUs regarding that object.

Class specific rules also exist which determine when other objects must be deleted because of dependencies.

```
constant deleteObjectRetransmitTime 300        -- seconds
constant maxDeleteObjectsCount 120

type ObjectIDWorldStateIDPair sequence {

    objectID                     ObjectID,
    worldStateID                 ObjectID

}

type DeleteObjectsVariant sequence {

    deletingSimulator            SimulationAddress,

    objectCount                  UnsignedInteger (8),
                                 unused (24),
    objects                      array (objectCount) of
                                   ObjectIDWorldStateIDPair

}
```

9.3.10 Set World State PDU

To set the current World State (see Section 9.1.4 [World State], page 49), the simulator must issue a Set World State PDU. This PDU should be retransmitted every 10 seconds, for as long as the simulator wishes to enforce this world state. If two simulators disagree about what the current world state should be, the state of the world may toggle between two frames. This condition is detectable (during the duration in which a simulator wishes to set the world state, it should not hear Set World State PDUs from other simulators), so the application can remedy the situation if necessary.

A series of different Set World State PDUs can be sent out at arbitrary intervals, for an animation effect.

A simulator may choose to ignore this PDU. Doing so merely means that the user does not want to see animation being controlled by another simulator. Similarly, a simulator may animate privately without issuing this PDU. However, if objects are to be projected via the Simulation Protocol, this private behavior may be confusing.

Upon receipt of a Set World State PDU, at the simulator's discretion, the simulator will adjust the state of all displayed objects to the version of each object correct for the specified world state. If the world state is not known to the simulator, it should not change the state of any objects. The simulator may also set a simulated clock to the time specified in the PDU, and increment the clock according to the factor identified in the PDU (this clock is for display only, subsequent world states should not be triggered until a different Set World State PDU is received).

A clock rate of +0.0 or -0.0 is used to indicate that simulated clocks on other machines should not be incremented over time (such as when the user pauses the animation). After receiving a setWorldStatePDU with a non-zero clock rate, and then not receiving any for 24 seconds, the simulator should stop incrementing its clock.

The worldState field should refer to an object of class World State.

```
constant setWorldStateTransmitTime 10          -- seconds
constant setWorldStateTimeoutTime 24           -- seconds

type SetWorldStateVariant sequence {

    requestingSimulator        SimulationAddress,

    -- (Simulated time) / (real time) clock factor
    clockRate                  Float (32),

    -- Current clock time
    secondsSince1970           UnsignedInteger (32),

    worldState                 ObjectID,
                               unused (16)
}
```

9.3.11 Nomination PDU

When a simulator is determined to have disappeared from the network (no Simulator Present PDUs have been heard for 48 seconds), each simulator will compare its own stated load (the one

transmitted in its own last Simulator Present PDU) with the load of other simulators on the network. If its load is lower than the others, the simulator will immediately take ownership of all overlay objects owned by the missing simulator. If its load is not the lowest, it must issue a Nomination PDU identifying the simulator which should assume control for the missing simulator. It is possible that two different simulators will not nominate the same simulator, however, the procedures used to determine ownership will resolve these conflicts.

Upon receipt of a Nomination PDU, a simulator should first confirm that it agrees the identified simulator is down (that it has not heard a Simulator Present PDU from that simulator within the last 48 seconds). If it does agree, the simulator should change the ownership field and increment the sequence number of every object owned by the missing simulator, immediately broadcast the new information, and then rebroadcast each object every 30 seconds thereafter. If it does not agree, it should ignore the request unless the nominating simulator is the simulator identified as missing. In that case, the nominated simulator may assume ownership of the objects at its discretion. This way, a simulator which detects an overload condition can ask for help with its packet handling requirements. Also, a simulator which is brought down intentionally (by hitting quit, for example) can use this PDU to facilitate orderly transition of object ownership.

A nominated simulator should establish ownership of these objects before processing further nomination PDUs. Doing so will lead to redundant nomination having little effect on performance.

It is unlikely that a simulator which missed the last few Simulator Present PDUs from a living simulator will also choose itself as the least loaded simulator. Under normal circumstances, the nominated simulator will know that the missing simulator is actually present. However, if a bad timeout does occur, the only consequence is that the errant simulator will unnecessary take over objects from a live simulator on the network.

It is possible that when a simulator goes down, the objects owned by that simulator will be lost, although it is unlikely. For this to happen, multiple simulators would have to disagree as to who is least loaded (which can happen if they miss one another's Simulator Present PDUs), and would have to also miss each other's Nomination PDUs. Such catastrophic failure might result if network hardware is interrupted, but is unlikely otherwise.

```
type NominationVariant sequence {

    nominatedSimulator          SimulationAddress,
    nominatingSimulator         SimulationAddress,
    missingSimulator            SimulationAddress

}
```

9.3.12 Top Level PDU

```
type PersistentObjectPDUKind enum (8) {
    simulatorPresentPDUKind (1),
    describeObjectPDUKind (2),
    objectsPresentPDUKind (3),
    objectRequestPDUKind (4),
    deleteObjectsPDUKind (5),
    setWorldStatePDUKind (6),
    nominationPDUKind (7)
}

type PersistentObjectProtocolVersion enum (8) {
    persistentObjectProtocolVersionJan91 (1),
    persistentObjectProtocolVersionJul91 (2),
    persistentObjectProtocolVersionAug91 (3),
    persistentObjectProtocolVersionSep91 (4),
    persistentObjectProtocolVersionJuly92 (5),
    persistentObjectProtocolVersionNov92 (6),
    persistentObjectProtocolVersionDec92 (7),
    persistentObjectProtocolVersionJan93 (8)
}

type DatabaseID UnsignedInteger (8)

type PersistentObjectPDU sequence {

    version                     PersistentObjectProtocolVersion,
    kind                        PersistentObjectPDUKind,
    exercise                    ExerciseID,
    database                    DatabaseID,
                                unused (32),

    variant                     choice (kind) of {

        when (simulatorPresentPDUKind)
          simulatorPresent      SimulatorPresentVariant,

        when (describeObjectPDUKind)
          describeObject        DescribeObjectVariant,

        when (objectsPresentPDUKind)
          objectsPresent        ObjectsPresentVariant,

        when (objectRequestPDUKind)
```

```
          objectRequest         ObjectRequestVariant,

        when (deleteObjectsPDUKind)
          deleteObjects         DeleteObjectsVariant,

        when (setWorldStatePDUKind)
          setWorldState         SetWorldStateVariant,

        when (nominationPDUKind)
          nomination            NominationVariant

    }
}
```

9.4 Throughput

The packet loss rate on a Mips Magnum platform at 2200 pps environment is 1:1500. This is not a significant loss rate (0.06%), so the maximum packet handling capacity of a Magnum is something larger than 2200 pps. (2200 pps is the maximum output rate of a Magnum sending 100 byte packets). Assuming that a very small proportion of the packets will have to be copied from the Lance descriptors to the rings, the raw packet handling rate of the machine is a good indication of the number of objects which can be supported on the net. We can make this assumption because the determination that a packet is redundant to information currently stored in memory (the normal case) is a trivial comparison that can easily be done before the packet is copied into the rings (similar to the distance checks done on the CMC card in a tank simulator). Since an object requires retransmission every 30 seconds, this leads to a maximum capacity of 66,000 objects, with the odds of missing a packet regarding a particular object 1 in 1500, and the odds of missing two packets regarding the same object (and hence timing it out) are 1 in 22,500,000.

The packet loss rates will be higher when most of the packets received cannot be filtered (such as when a large scenario is loaded, and hence many new objects are being created at once). However, these peak load conditions should be rare, and the redundancy of the protocol should compensate as soon as the network reverts to its normal state.

This throughput rating is a hardware-only evaluation. A larger limiter on throughput may be the applications' ability to transmit PDUs and remain responsive to the user.

This protocol may be used in conjunction with the Simulation protocol, which could consume as much as 1000 of the 2200 pps which the Lance can handle. This would reduce the throughput to 36,000 objects in a Warex size exercise.

The application level need for objects is relatively unbounded. For example, a theater-level operation including detailed information about 6000 objects, with 1500 of those objects moving from one world state (see Section 9.1.4 [World State], page 49) to another, yields only 40 world states which may be maintained at once (three weeks, assuming 12 hour updates).

Function Index

Q

S

W

Index

Q

S

T

APPENDIX C7: VIDS PROTOCOL EXTENSION

TBS